VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV INFORMATIKY
DEPARTMENT OF INFORMATICS

NÁVRH ŘEŠENÍ PRO EFEKTIVNÍ ANALÝZU BEZPEČNOSTNÍCH DAT

DESIGN OF A SOLUTION FOR EFFECTIVE ANALYSIS OF SECURITY DATA

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. ŠIMON PODLESNÝ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. JAN LUHAN, Ph.D., MSc**
SUPERVISOR

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav informatiky
Student:	**Bc. Šimon Podlesný**
Studijní program:	Systémové inženýrství a informatika
Studijní obor:	Informační management
Vedoucí práce:	**Ing. Jan Luhan, Ph.D., MSc**
Akademický rok:	2020/21

Ředitel ústavu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Návrh řešení pro efektivní analýzu bezpečnostních dat

Charakteristika problematiky úkolu:

Úvod
Cíle práce, metody a postupy zpracování
Teoretická východiska práce
Analýza současného stavu
Vlastní návrhy řešení
Závěr
Seznam použité literatury
Přílohy

Cíle, kterých má být dosaženo:

Navrhnout vhodnou architekturu v oblasti velkých dat a bezpečnosti pro efektivní práci se zaměřením na analýzu úniku hesel.

Základní literární prameny:

LABERGE, R. Datové sklady - Agilní metody a business intelligence. Praha: Computer Press, 2012. 352 s. ISBN 978-80-251-3729-1.

SAKR, S. and A. ZOMAYA. Encyklopedia of Big Data Technologies. 1st ed. Springer International Publishing, 2019. 1820 p. ISBN 978-3-319-77526-5.

SHARDA, R., D. DELEN and E. TURBAN. Business Intelligence, Analytics and Data Science: A Managerial Perspective. 4th ed. UK: Pearson, 2017. 512 p. ISBN 978-1-292-22054-3.

STROHBACH, M., J. DAUBERT, H. RAVKIN and M. LISCHKA. Big Data Storage. In New Horizons for a Data-Driven Economy. Springer: 2016. Dostupné z: https://doi.org/10.1007/978-3-319-21569-3_7. ISBN 978-3-319-21569-3.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2020/21

V Brně dne 28.2.2021

L. S.

Mgr. Veronika Novotná, Ph.D.
ředitel

doc. Ing. Vojtěch Bartoš, Ph.D.
děkan

Abstrakt

Cieľom práce je navrhnúť vhodnú architektúru v oblasti spracovania Big data a to so zameraním na analýzu uniknutých hesiel. Za týmto účelom boli popísané a porovnané úložné systémy a vytvorený návrh architektúry, ktorá dokáže načítať, spracovať, uložiť a sprístupniť dáta pre analýzu, berúc do úvahy faktory ako autentifikácia a autorizácia a princípy návrhu modernej agilnej infraštruktúry.

Abstract

The goal of this thesis is to design architecture capable of processing big data with focus on data leaks. For this purpose multiple data storage systems were described a and compared. The proposed architecture can load, process, store and access data for analytic purposes while taking into account authentication and authorisation of users and principles of modern agile infrastructure.

Kľúčové slová

Big data, Apache NiFi, MongoDB, návrh, architektúra, spracovanie, uniknuté údaje

Keywords

Big data, Apache NiFi, MongoDB, design, architecture, processing, data leaks

Citácia

PODLESNÝ, Šimon. *Návrh řešení pro efektivní analýzu bezpečnostních dat*. Brno, 2021. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská. Vedoucí práce Ing. Jan Luhan, Ph.D., MSc

Prehlásenie

Prohlašuji, že předložená diplomová práce je původní a zpracoval jsem ji samostatně. Prohlašuji, že citace použitých pramenů je úplná, že jsem ve své práci neporušil autorská práva (ve smyslu Zákona č. 121/2000 Sb., o právu autorském a o právech souvisejících s právem autorským).

. . . . . .

Šimon Podlesný

15. mája 2021

Poďakovanie

V prvom rade by som sa chcel poďakovať vedúcemu práce pánovi doktorovi Jánovi Luhanovi za odborné vedenie práce a za konzultácie, na ktoré si vždy našiel čas a to aj napriek veľkému pracovnému zaťaženiu. Ďalej by som sa chcel poďakovať konateľovi spoločnosti REDAMP SECURITY s.r.o., pánovi doktorovi Dominikovi Malčíkovi, za materiálnu a technickú podporu pri vypracovaní diplomovej práce. Vďaka patrí taktiež priateľke Kataríne Muškovej za pomoc pri štylizovaní a formátovaní práce a v neposlednom rade rodine a známym, ktorí ma podporovali pri jej vypracovaní. Špeciálne by som chcel poďakovať taktiež Alexandre Elbakyan a to za službu Sci-Hub, bez ktorej podpory by vypracovanie práce v čase globálnej pandémie COVID-19 bolo oveľa viac komplikované.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Práca sa zaoberá oblasťou spracovania Big data, ktorá sa líši od klasického návrhu architektúry a to hlavne požadovaným miestom a výkonom, ktorý musí byť ľahko rozšíriteľný na viacero fyzických zariadení. Zameranie práce je na oblasť počítačovej bezpečnosti, so špecifickým zameraním na analýzu uniknutých údajov. Pri práci sa spolupracovalo so spoločnosťou REDAMP SECURITY s.r.o., pre ktorú je samotný návrh vytvorený.

Práca je rozčlenená na niekoľko základných kapitol. V Kapitole 2 sú detailne rozpísané ciele práce, v podobe požiadaviek, ktoré by mala výsledná architektúra spĺňať a s ktorých sa vychádzalo pri vypracovávaní tejto práce.

V Kapitole 3 je následné popísaná teória k jednotlivým technológiám, s ktorými sa v práci pracuje. Špeciálna pozornosť sa venovala hlavne technológiám, ktoré sú použité vo finálnom návrhu, ale je tu presah aj do širšieho ponímania analýzy Big data, vrátane popísania technológie, akou je Hadool a súborového systému HDFS, ktorý sa v práci síce priamo nevyužíva, ale tvorí základný stavebný kameň celej architektúry. Sú tu zároveň spomenuté a popísané technológie, ktoré sa vo finálnom návrhu síce nepoužili, ale za to majú čiastočný presah, alebo ich je možné použiť ako alternatívu v prípade zmien na strane návrhu. Zároveň sú tu popísané aj niektoré technológie, ktoré sú v príliš skorej fáze na nasadenie (MyRocks, Zed store), ale ich vývoj je vhodné sledovať do budúcna. Teoretické východiská boli výzvou, nakoľko mnohé z technológie sú príliš mladé na to, aby o nich bolo k dispozícii veľké množstvo údajov a článkov. Ako príklad sa môže uviesť projekt Apache Solr, ktorý je pod aktuálnou značkou iba od februára roku 2021. Z toho dôvodu boli zdrojom informácii o daných technológiách často webové stránky danej technológie.

Základ práce tvorí Kapitola 4, ktorá je venovaná analýze súčasného stavu. Sú v nej popísané existujúce riešenia v podobe služieb, ktoré majú podobnú funkčnosť a funkcionalitu. Sú popísané

dôvody, prečo takúto analýzu robiť a následne sú detailne rozpísané parametre údajov, s ktorými sa bude pracovať v prípade nasadenia navrhnutej architektúry. Detailne sú analyzované hlavne dve základné, verejne dostupné kolekcie hesiel. Tieto kolekcie hesiel (desaťtisíce textových súborov s menami, emailami a heslami) slúžili ako hlavný podklad pre popísanie možných spôsobov spracovania údajov. Následne tieto údaje boli použité ako vstupné dáta pre úložné systémy MongoDB, Greenplum a PostgreSQL na ich porovnanie a to ako z pohľadu výkonu, silných a slabých stránok, tak aj z pohľadu miesta, ktoré výsledná databáza zaberie.

Na základe informácii z analytickej časti sa pristúpilo k navrhnutiu vlastnej architektúry v Kapitole 5 a to vrátane čiastočnej implementácie, ktorá však bola robená nad rámec zadania. Sú popísané a demonštrované spôsoby automatizovaného a poloautomatizovaného spôsobu spracovania vstupných údajov a spôsobu ich uloženia do databázy. Sú navrhnuté spôsoby, ako overiť validitu týchto údajov a ako k uloženým dátam pristupovať v rámci potrebnej práce s dátami. Na demonštračné účely bola vytvorená aplikácia, ktorá umožňuje dáta prehľadávať a pracovať s nimi cez jednoduché webové rozhranie. Zároveň je vytvorený návrh, ako k týmto dátam pristupovať a exportovať ich pre prípadne zmeny na úrovni architektúry, alebo potreby exportu čiastočne anonymizovaných údajov. Výsledné riešenie je zhrnuté v poslednej časti danej kapitoly.

Kapitola 2

Ciele práce, metódy a postupy spracovania

Dátová analýza bezpečnostných dát je komplexnou úlohou, kde sa pod spoločnou témou spojuje niekoľko problematických častí. K dispozícii je veľké množstvo neštrukturovaných alebo čiastočne štrukturovaných údajov rôzneho typu. Týchto údajov je dostatočne veľké množstvo na to, aby ich manuálne spracovanie z neštrukturovanej do štrukturovanej podoby nebolo možné. Zároveň je údajov dostatočne veľké množstvo na to, aby bežný Relational Database Management System (RDBMS) systém nebol použiteľný a to ako z dôvodov rýchlosti, tak aj z dôvodu potrebného miesta na uloženie. Naskytá sa teda otázka, ako najlepšie navrhnúť systém, ktorý:

- Nebude obmedzený fyzickou veľkosťou úložného priestoru
- Bude škálovateľný
- Umožní paralelné spracovanie a prístup
- Nebude predpokladať žiadne relácie medzi údajmi
- Nebude predpokladať počet stĺpcov, veľkosť a dátový typ údajov
- Umožní prístup analytikom, ideálne v podobe jazyka SQL
- Umožní prístup cez Application Programming Interface (API) na prepojenie s ďalšími systémami
- Dáta budú prístupne aj pre nezaškolených užívateľov v podobe webového rozhrania

- Uchová čo najväčšie množstvo metainformácií
- Výsledok sa bude dať nazvať za *data lake* (nie *data swamp*)

Tieto požiadavky môžu znieť na prvý pohľad utopisticky. Môže sa zdať, že niektoré požiadavky si na prvý pohľad protirečia a to hlavne z pohľadu rozšíreného pri relačných databázach. Treba však zahodiť zastaralý pohľad na dáta ako na niečo, čo sa dá dokonale roztriediť, zaškatuľkovať a vytvoriť medzi údajmi dokonalé väzby.

Hlavne zameranie práce je navrhnutie architektúry, ktorá tieto parametre definované vyššie spĺňa. Vedľajším zameraním práce je však poukázať na to, že klasický prístup využívania relačných databáz na spracovanie veľkého množstva údajov, ktoré sú často nekonzistentné a neštrukturované je chybný.

K tomuto účelu bolo spracovaných a upravených 3,2 miliardy záznamov z verejne dostupných kolekcii prístupových údajov, ktoré sa používali ako vstupné údaje pre výkonnostné porovnanie rôznych úložných technológii (`data storage`).

Po získaní potrebných údajov, parametrov, silných a slabých stránok jednotlivých technológii sa následne pristúpilo k vypracovaniu finálneho návrhu. Ten je rozčlenený na niekoľko častí a následne zlúčený a zhrnutý v poslednej časti danej kapitoly.

Kapitola 3

Teória práce

V kapitole Teória práce sú popísané teoretické východiská. Oblasť spracovania Big data je široká a z toho dôvodu boli popísané hlavne technológie a princípy za technológiami, ktoré sa priamo alebo nepriamo budú využívať v návrhu architektúry.

3.1 Problematika Big data

Masívna kolekcia dát vedie k rôznym výzvam v oblasti počítačovej vedy. Denne sa nahrávajú petabajty dát [47]. Predtým, ako vôbec môžme uvažovať o ich využití, musíme uvažovať o tom, ako ich uložiť. Aký ekosystém aplikácii potrebujeme vytvoriť, aby sme dáta mohli získať a analyzovať?

Ako sa ukáže v nasledujúcej práci, mnohé moderné aplikácie vyžadujú prácu s dátami tak veľkými, že tradičný prístup k návrhu a práci s dátami nie je efektívny a často ani použiteľný [51].

Tradičné modely analýzy často vyžadujú prácu s informáciami v rámci pamäte Random Access Memory (RAM) a ich časová náročnosť je presne vypočítaná z počtu inštrukcii, ktoré sa musia vykonať. Sociálne siete, vyhľadávače obsahu, ale aj online obchody ako Amazon, však často pracujú s dátami, pri ktorých tieto nároky nie je možné technický splniť. [44]

Obecne je možné povedať, že problém spracovania Big data je možné rozdeliť na tri hlavné problémy:

- Objem, ktorý je potrebné spracovať
- Štruktúra dát
- Náročnosť na rýchlosť spracovania

V prípade problémov s objemom dát, ktoré je potrebné spracovať, môžme zobrať ako príklad sieťovú sondu. V prípade zaznamenania plnej dátovej prevádzky (full packet capture) je potrebné spracovať nad gigabitovou linkou až 125MB/s dát. V prípade viacerých sond na rôznych sieťach je množstvo denne zaznamenaných dát v jednotkách terabajtov. Zároveň je tu podmienka real-time spracovania, nakoľko úplne zaznamenanie komunikácie neprichádza do úvahy. Táto oblasť Big data sa nazýva flow processing a využíva sa okrem iného aj pri strojovom učení [49].

Problematická štruktúra dát spočíva prevažne vo veľkom množstve položiek, nad ktorými je potrebné robiť dátovú analytiku. Ostať pri analógii zo sondou, tak sa jedná o analýzu už spracovaných dát. Ako príklad sa môže uviesť využitie Elastic Logstash Kibana (ELK) haldy (stacku) v prípade Suricata Intrusion Detection System (IDS). Grafické rozhranie nástroja Kibana môže indexovať a pracovať s viac ako tisíc indexmi zároveň. V prípade klasického RDBMS prístupu by len samotná veľkosť indexov dokázala prekonať veľkosť uložených dát.

Náročnosť na rýchlosť zase vyplýva z potreby získať výsledné informácie v potrebnom čase. V prípade našej analógie z IDS systémom je potrebné získať dáta v dostatočne rozumnom čase, aby spojenie so serverom nebolo uzavreté. To je často problematické pre RDBMS systémy, nakoľko časová náročnosť získania údajov rastie okrem iného aj z dôvodu využitia algoritmu B-tree, ako hlavného indexovacieho algoritmu vo väčšine moderných relačných databáz, pričom jeho časová náročnosť je logaritmická [53].

3.2 Data storage technológie

Nasledujúca kapitola sa zaoberá spôsobmi uloženia údajov na samotnom hardware. Technológia sa dá rozdeliť na dva základné prístupy a tým je klasický prístup, časom overenej Redundant Array of Inexpensive Disks (RAID) technológie, ktorá umožňuje ako navýšenie kapacity, tak i priepustnosti systému pomocou vhodných algoritmov na úrovni fyzického priestoru diskov. Druhým spôsobom je technológia, ktorú zastupuje Hadoop Distributed File System (HDFS), pri ktorej sa ruší klasické oddelenie úložného priestora a výpočetného modulu. Taktiež sa nepočíta s fyzickou konzistenciou dát, ale namiesto toho sú dáta viacnásobne duplikované na úrovni aplikačnej vrstvy. Vychádza sa z predpokladu, že úložný priestor je lacný a paralelné spracovanie dát je nevyhnutné.

RAID technológia

RAID je technológia virtuálizácie úložného priestoru, ktorá kombinuje viacero fyzických diskov do jedna až N logických diskov za účelom redundancie uložených dát, alebo vylepšenia výkonu [56]. Existuje niekoľko rôznych spôsobov rozloženia dát a metainformácií, ktoré sa bežne označujú ako RAID. Pre účely použitia v práci nás budú zaujímať nasledujúce:

- RAID 1
- RAID 0
- RAID 1+0 (RAID 10)
- RAID 5
- RAID 6

RAID 0 prekladá (*stripping*) dáta medzi viacero pevných diskov. Tým sa zväčší úložný priestor (ktorý sa počíta ako suma veľkosti všetkých diskov v RAID 0), zvyšuje sa rýchlosť zápisu a čiastočne aj rýchlosť čítania, nakoľko je možné paralelne čítať a zapisovať zároveň z viacerých diskov. Na druhú stranu sa však pri poruche jedného disku stanú všetky dáta nepoužiteľné (nakoľko teoreticky chýba polovica informácii v prípade využitia dvoch diskov).

Tento nedostatok rieši RAID 1, ktorý sa občas prezýva aj ako zrkadlenie diskov. Pri tejto RAID technológii sú všetky dáta zapísané na všetky disky zároveň, čo vytvára redundanciu pre prípadné poruchy. Výsledkom je výrazne zrýchlenie čítania z disku avšak za cenu spomalenia zápisu (dáta

sa musia zapisovať na všetky disky a sú obmedzené najpomalejším diskom). Nevýhodou je taktiež fakt, že veľkosť výsledného RAID poľa je definovaná kapacitou najmenšieho disku v RAID poli.

Kombináciou vyššie popísaných RAID technológii sa dostaneme k RAID 10 (občas zapísaného aj ako RAID 1+0), pri ktorom sú dáta zapisované prekladane a pritom sú zrkadlené. Nevýhodou je znížená kapacita, ktorá sa vypočíta ako $(n * c)/2$, pričom c je kapacita najmenšieho disku a n reprezentuje počet diskov v RAID poli. Výhodou je veľká prenosová rýchlosť a jednoduchá obnova disku pri výpadku. Z dôvodu vysokých prenosových rýchlosti je táto varianta prevažne používaná v Online Transaction Processing (OLTP) systémoch [62].

Je taktiež vhodné upozorniť ešte na existenciu technológie RAID 01, ktorá však nie je praktická a z toho dôvodu sa často nepoužíva.

V prípade RAID 5 a RAID 6 je používaná tzv. parita (samo-opravný kód). Pre RAID 5 potrebujeme na jeho prevádzku minimálne 3 fyzické disky, pričom kapacita jedného je využívaná na ukladanie parity. Táto parita je rozprestretá medzi všetkými diskami a v prípade výpadku jedného disku je možné jeho dáta obnoviť z ostatných dvoch diskov. Vďaka tomu sa v prípade troch diskov s rovnakou kapacitou vieme dostať ku kapacite až 75%, na rozdiel od 50% kapacity, ktorú by sme dosiahli pri RAID 10. Rýchlosť zápisu je nižšia, nakoľko je potrebné vypočítať paritný kód pri zápise, avšak rýchlosť čítania je vyššia, nakoľko je možné využiť paralelného prístupu k čítaniu dát.

Z dôvodu zvýšenej rýchlosti čítania a zníženej rýchlosti zápisu sa RAID 5 odporúča [62] prevažne pre nekritické Online Analytical Processing (OLAP) systémy.

V prípade kritických OLAP systémov a dlhodobého uloženia dát v OLAP systéme sa odporúča [62] využiť RAID 6, pri ktorom je potrebné použitie minimálne štyroch diskov a v rámci ktorého sú vypočítané dve parity rozdielnym spôsobom. Výhodou je odolnosť voči výpadku dvoch zo štyroch diskov. Rýchlosť čítania je zrovnateľná z RAID 5, ale zápis je pomalší z dôvodu dvojnásobného počtu parít. Tým sa kapacitou blížime k RAID 10, ktorý je však rýchlejší na zápis, nakoľko nemusí vypočítavať paritu. Na rozdiel od RAID 1 je možné RAID 6 využiť pre 5+ diskov, kedy kapacita nie je polovičná a stúpa s počtom diskov.

RAID 5 a RAID 6 ponúkajú viac ochrany ako RAID 0 s nižšou cenou za uložený gigabajt ako RAID 10, ktorý vyžaduje dvojnásobnú kapacitu voči želanej [62] aj keď za cenu zníženej rýchlosti zápisu voči RAID 10, čo však nie je problém pri OLAP systémoch, ktoré sú z veľkej časti zamerané na čítanie údajov namiesto ich zápisu a modifikácie.

HDFS

Jadrom Apache Hadoop ekosystému (popísaného v sekcii 3.3) je ukladacia časť, známa ako HDFS. HDFS rozdeľuje veľmi veľké súbory (vrátane súborov s veľkosťou nad 1GB) na bloky, ktoré sú rozdistribuované medzi viacero uzlov (nodes) v clusteri. Spoľahlivosť je zabezpečená replikáciou blokov medzi viacero uzlov (v základných nastaveniach 3). Hadoop architektúra následne prostredníctvom algoritmu MapReduce rozdistribuuje kód medzi uzly, na ktorých sa vykoná samostatná operácia.

Tento prístup využíva dostupnosti dát, kde uzly pracujú s iba lokálne dostupnými dátami. To umožňuje rýchlejšie a efektívnejšie spracovanie veľkého množstva dát oproti klasickému prístupu v superpočítačoch, kde existuje paralelný sieťový súborový systém (NFS/SAMBA) a výpočtový uzol, pričom sú prepojené cez vysoko-rýchlostnú sieť. **Hadoop umožňuje lacnejšiu prevádzku a vyšší výkon bez potreby špecifického hardware.** Taktiež je možné HDFS využívať ako vo vlastnom dátovom centre, tak aj v cloudovej infraštruktúre aktuálnej najväčších poskytovateľov (Microsoft, IBM, Google, Amazon a Oracle) formou služby. [59]

3.3 Apache Hadoop

Aj keď v samotnej práci sa Apache Hadoop priamo nevyužíva (s výnimkou formátu Apache Avro), je okolo neho vytvorený celý ekosystém aplikácii z prostredia nadácie Apache, ktoré vo výslednom návrhu budú čiastočne alebo úplne využité. Z toho dôvodu je vhodné spomenúť si aj tento základný kameň, okolo ktorého je vybudovaná infraštruktúra a ktorý si našiel svoje uplatnenie u veľkých spoločností akými je napríklad Netflix [38].

Apache Hadoop je softvérová knižnica v podobe frameworku, ktorá umožňuje distribuované spracovanie veľkého množstva dát v rámci clusteru počítačov, využívajúc pritom jednoduché programovacie modely. Je navrhnutý tak, aby sa dal škálovať z jedného serveru na tisíce serverov, pričom každý ponúka lokálne úložisko a výpočtový výkon.

Radšej ako sa spoliehať na hardware, ktorý by zabezpečoval vysokú dostupnosť (viď RAID), tak už samotná knižnica je navrhnutá spôsobom, aby vedela detegovať a riešiť zlyhania na úrovni aplikačnej vrstvy. Tým je možné zabezpečiť službu s vysokou dostupnosťou a to na clusteri zariadení, ktoré nemusia byť samostatne spoľahlivé. [6]

Obr. 3.1: Logo Hadoop HDFS [6]

Apache Spark

Nasledujúca sekcia čerpá informácie z oficiálnych webových stránok projektu Apache Spark [8].

Apache Spark je rýchly a univerzálny engine[1] na spracovanie informácii, ktorý je kompatibilný s HDFS. Umožňuje beh v Hadoop clusteri cez YARN[2], alebo samostatne. Okrem HDFS umožňuje taktiež spracovanie dát z dátových úložísk HBase, Cassandra, Apache Hive alebo iného Hadoop vstupného formátu. Je navrhnutý ako na dávkové spracovanie (podobné MapReduce v Apache Hadoop), tak aj na spracovanie dátových tokov (streamov), interaktívnych dotazov (queries) a strojové učenie.

O Apache Spark sa dá hovoriť ako o unifikovanom analytickom nástroji, ktorý umožňuje rozsiahle spracovanie dát z rôznych zdrojov. Medzi podporované jazyky patrí: Java, Scala, Python,

[1] Slovo engine znamená motor, ktorý poháňa daný typ aplikácie. V technickej praxi sa slovo neprekladá.

[2] V tomto prípade nehovoríme o balíčkovacom systéme YARN, ale o Apache Hadoop YARN, čo je manažér zdrojov a úloh v infraštruktúre Hadoop

R a SQL. Spark ponúka viac ako 80 vysokoúrovňových operátorov, ktoré umožňujú vybudovanie paralelných aplikácii a ktoré je možné využiť interaktívne v jazykoch popísaných vyššie.

Apache Spark sa často skloňuje aj ako náhrada za Apache Hadoop, hlavne vďaka jeho vyššiemu výkonu. Nástroje však nie je možné úplne porovnávať jedna k jednej, nakoľko Apache Spark zvolil rozdielny prístup v niektorých oblastiach a má širšie uplatnenie. Ich detailné porovnanie by však bolo nad rozsah tejto práce.

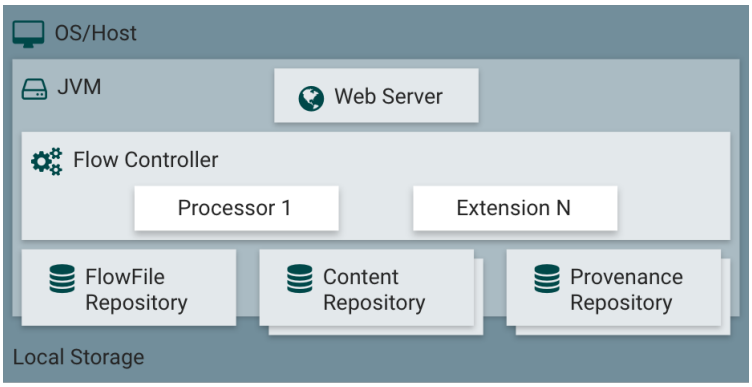
Obr. 3.2: Logo projektu Apache Spark [8]

Apache NiFi

Apache NiFi je open-source nástroj pre automatizáciu toku dát medzi rôznymi systémami. Využíva konceptu Extract Transform Load (ETL) a je založený na nástroji NiagaraFiles (z čoho vyplýva názov Apache NiFi), ktorý bol pôvodne vyvíjaný americkou spravodajskou agentúrou NSA a ktorý bol uvoľnený v rámci technologického transferu NSA v roku 2014. [29]

Nástroj Apache NiFi bol vytvorený na automatizovanie toku dát medzi systémami. Termín dátový tok (*dataflow*) je používaný v rôznych kontextoch. V prípade NiFi sa využíva na automatizované manažovanie dátových tokov medzi informačnými systémami. Tento problém (dátových tokov medzi informačnými systémami (IS)) existuje už od časov, kedy spoločnosti mali viac ako jeden podnikový systém, pričom niektorý systém dáta produkoval a iný ich pre zmenu potreboval konzumovať. [7]

Architektúra aplikácie sa skladá z niekoľkých vrstiev. Ako je možné vidieť na obrázku 3.3, na vrchole je webové rozhranie, cez ktoré sa aplikácia ovláda. V rámci jedného kontroléra beží niekoľko FlowFile Procesorov, ktoré plnia rôznorodé účely a niekoľko Rozšírení (*Extensions*), ktoré je možné využiť na rozšírenie fungovania NiFi o vlastnú funkcionalitu (vrátanie programovania vlastných Procesorov).



Obr. 3.3: NiFi architektúra [7]

FlowFile procesor (ďalej ako "procesor"), je základnou jednotkou spracovania informácii v rámci architektúry Apache NiFi. Procesor je abstraktnou jednotkou, reprezentujúcou akúkoľvek operáciu s dátami (FlowFile) alebo jeho metainfomáciami (atribútami). V čase písania práce Apache NiFi oficiálne podporovala 283 rôznych procesorov, pričom v prípade ich nedostupnosti je možné využiť API rozhranie na dopísanie vlastného procesora. Procesory sú rôzneho druhu. Existujú procesory, ktoré riešia pripojenie na API rozhranie sociálnych sietí ako napríklad Twitter-u [18],

umožňujú konverziu medzi formátami (ConvertAvroToJSON [12]), prácu s textom (SplitText [37], ExtractText [14]), ukladanie a čítanie dát z rôznych úložných systémov (SQL [35], MongoDB [17], FTP [16]), odosielanie emailov (PutEmail [34]), dotazovanie sa na AWS a Azure služby (ConsumeAzureEventHub [11], InvokeAWSGatewayApi [23]) alebo preklad textu (Yandex translate [41]).

Základnou jednotkou v aplikácii Apache NiFi je tzv. FlowFile. FlowFile reprezentuje abstraktnú jednotku informácie v rámci aplikácie. Ako príklad môžeme uviesť procesor SplitText. Na vstupe sa nachádza jeden FlowFile, reprezentujúci súbor, ktorý obsahuje tisíc riadkov. Procesor SplitText je nastavený tak, aby každý FlowFile, ktorý dostane na vstupe, rozdelil po jednom riadku (tieto parametre mu boli určené). V našom prípade sa teda jeden FlowFile príjme na vstupe a na výstupe získame tisíc FlowFiles, pričom každý FlowFile obsahuje jeden riadok z pôvodného súboru a atribúty, ktoré ho popisujú a ktoré získal buď formou dedenia z pôvodného FlowFile, alebo mu ich pridelil samotný procesor.

FlowFile Repository je miestom, kde sú fyzicky uložené metainformácie o FlowFiles. Na úrovni implementácie je to Write Ahead Log (WAL), ktorý sa využíva aj v prípade relačných databáz. V prípade napríklad výpadku napájania je možné využiť informácie, obsiahnuté vo WAL na obnovu konzistentného stavu systému.

Samotné dáta (FlowFiles) sú uložené v Content Repository. Content repozitárov môže byť špecifikovaných 1 až N a tým sa dá predísť úzkemu hrdlu (*bottleneck*) v podobe obmedzenia maximálnej rýchlosti čítania a zápisu na disk.

Informácie o živote FlowFile (*data provenance*) sú uložené v Provenance Repository. Vďaka tomu je možné sledovať životný cyklus informácie v rámci systému.

NiFi taktiež umožňuje od verzie 1.0 (aktuálne 1.13.2) paralelizáciu vykonávania úloh a to prostredníctvom aplikácie Apache Zookeeper. V rámci clustera si Zookeeper automatický určí koordinátora (mastera), ktorému všetky ostatné uzly automaticky nahlásia svoj status a stav. Následne sú všetky úlohy robené nad všetkými uzlami, ale vždy nad rozdielnou časťou dát. V rámci užívateľského rozhrania je možné pristupovať k ľubovolnému uzlu. Všetky zmeny sa automatický prenesú na ostatné uzly.

Okrem týchto základných komponentov v práci bude využívaná ešte pokročilejšia časť, ktorá umožní spracovanie veľkého objemu dát a to Flow Controller. Flow Controller funguje ako prostredník (*broker*) medzi procesmi a zabezpečuje výmenu FlowFile-ov. Tým sa obchádza využitie fronty (*queue*) pre daný proces, čo umožní zvýšiť priepustnosť spojenia.

Obr. 3.4: Logo aplikácie Apache NiFi [7]

Apache AirFlow

Apache Airflow je open-source projekt, ktorý pôvodne vznikol v spoločnosti Airbnb. Jedná sa o platformu, ktorá vznikla za účelom plánovania, monitorovania a spúšťania programovateľných tokov informácii (*workflow*).

Airflow má modulárnu architektúru a používa frontu správ (*message queue*) na manažovanie ľubovolného počtu pracovníkov (*workers*), čo umožňuje neobmedzené škálovanie.

Toky dát (*pipelines*) sú definované v jazyku Python, čo umožňuje ich dynamické generovanie a písanie kódu, ktorý vytvorí potrebný dátový tok dynamicky podľa potreby. Zároveň sú dátové toky od návrhu štíhle (*lean*)[3] a explicitné. Parametrizácia je zabudovaná priamo v jadre Apache Airflow a to prostredníctvom šablónovacieho enginu Jinja.

Aplikácia je zároveň jednoducho rozšíriteľná a umožňuje definovanie vlastných operátorov a rozširujúcich knižníc na vytvorenie úrovne abstrakcie, ktorá vyhovuje potrebám pracovného prostredia. Je možné využiť vlastností jazyka Python pre vytvorenie vlastného workflow a to okrem iného aj na plánovanie, cyklenie a dynamické generovanie úloh.

Na monitorovanie je k dispozícii robustné moderné webové rozhranie, bez nutnosti učenia sa formátov a rozhraní, akým je napríklad známa linuxová aplikácia CRON. K dispozícii sú taktiež informácie ohľadom stavu, výstupu a priebehu aktuálne spustených úloh.

K dispozícii sú tiež operátory, ktoré umožnia spustenie naplánovaných úloh v cludových službách typu Google Cloud Platform, Amazon Web Services, Microsoft Azure a iných. To umožňuje jednoduchú integráciu a rozšírenie existujúcej infraštruktúry o cloudové služby.

Uplatnenie Apache Airflow je široké a je možné ho využiť okrem iného aj na tvorbu modelov pre strojové učenie, manažovanie infraštruktúry v prípade Infrastructure as Code (IaC), transfer údajov a ďalšie technické úlohy, ktoré sa bežne vyskytujú v modernej IT spoločnosti. [30]

[3] Myslené z pohľadu systémových zdrojov na samotný chod

Obr. 3.5: Logo projektu Apache Airflow [30]

Apache Drill

S exponenciálnym rastom dát v posledných rokoch a prechodom na rýchly vývoj aplikácii sú čím ďalej tým častejšie dáta ukladané v nerelačných databázových systémoch typu Hadoop a NoSQL, alebo sú uložené v cloudových riešeniach. Apache Drill umožňuje analytikom, firemným užívateľom, výskumníkom a vývojárom prehľadávanie a analýzu dát, bez obetovania flexibility a agilnosti, ktoré tieto riešenia ponúkajú. Apache Drill spracováva dáta in-situ, a teda bez toho, aby užívateľ musel definovať schémy alebo dáta transformovať.

Apache Drill je inovatívny distribuovaný SQL engine, navrhnutý tak aby dokázal prehľadávať a robiť analýzu nad nerelačnými databázovými technológiami. Užívatelia môžu robiť dotazy nad dátami pomocou bežnej SQL syntaxe a BI nástrojov a to bez potreby vytvárania a manažmentu schém.

Vďaka tomu, že nástroj má plnú podporu SQL, užívatelia môžu využívať bežné nástroje na dátovú analytiku a Business Intelligence (BI) akými sú napríklad:

- Tableau
- Qlik
- MicroStrategy
- Spotfire
- SAS
- Microsoft Excel

Podpora Java Database Connectivity (JDBC) a Open Database Connectivity (ODBC) ovládačov umožňuje využiť Apache Drill taktiež na prepojenie s bežnými relačnými databázovými systémami akými sú napríklad Oracle, MySQL, MariaDB alebo PostgreSQL. Podpora Representational state transfer Application Programming Interface (REST API) rozhrania zase umožňuje vývojárom písanie vlastných aplikácii na tvorbu vizualizácii.

Vďaka Apache Drill virtuálnej množine údajov (*datasets*) je možné aj tie najkomplexnejšie nerelačné dáta mapovať do BI prívetivejšej štruktúry, ktorá umožní užívateľom prehľadávať a vizualizovať dáta v nástrojoch podľa ich želania. [31]

Čo sa týka podpory NoSQL databázových a súborových systémov, tá je nasledujúca:

- HBase
- MongoDB
- MapR-DB
- HDFS
- MapR-FS
- Amazon S3
- Azure Blob Storage
- Google Cloud Storage
- Swift
- NAS
- lokálne súbory

Prostredníctvom Apache Drill je možné spojiť napríklad užívateľský profil uložený v MongoDB s priečinkom užívateľských logov uložených v Hadoop.

Obr. 3.6: Logo technológie Apache Drill [31]

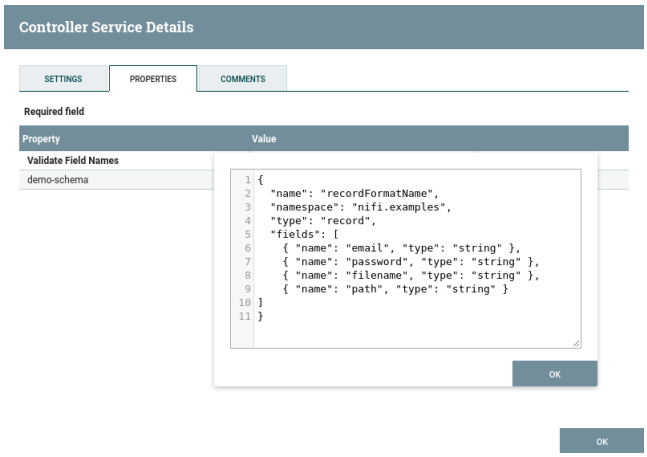
Apache Avro

Avro je riadkovo orientovaný úložný formát, ktorý je taktiež popisovaný ako dátový serializačný systém podobný Java serializácii. Avro poskytuje bohaté dátové štruktúry, kompaktný, rýchly binárny dátový formát v podobe kontajnera, ktorý je možné využiť na uloženie trvácnych dát s možnosťou volania cez remote procedure call (RPC). Nie je vyžadované generovanie žiadneho kódu pre čítanie a zápis dátových súborov, podobne ako nie je nutné využiť alebo implementovať RPC protokol.

Alternatívne systémy v podobe Java serializácie, Thrift a Protocol Buffers pracuju jedine počas behu programu. Zároveň Avro umožňuje poskytnutie optimalizovanejšieho výkonu za behu prog-

ramu. Na druhú stranu Avro sa spolieha na definovanú schému (možné vidieť na obrázku 3.7). V schéme je definovaná štruktúra dát a jej definícia je použitá pre čítanie a zápis dát.

Avro je taktiež možné použiť na uloženie veľkého množstva malých súborov do jedného Avro súboru v rámci HDFS a to za účelom redukovania využitia namenode pamäte v systéme. [57]



Obr. 3.7: Definovanie štruktúry pre prácu s formátom Avro v nástroji Apache NiFi [58]

Apache Solr

Apache Solr je populárna, výkonná, open-source vyhľadávacia platforma vybudovaná nad projektom Apache Lucene. Projekt vznikol v roku 2006 ako podprojekt Apache Lucene a osamostatnil sa v roku 2021 ako Top Level Projekt (TLP) nadácie Apache.

Platforma je optimalizovaná na veľké objemy dát a veľký objem simultánnych prístupov, pričom dokáže pracovať ako so schémovými, tak aj bezschémovými údajmi. Má podporu jednoduchého definovania dátových typov a štruktúr.

Medzi jeho hlavné vlastnosti patrí podpora full-textového vyhľadávania, zvýrazňovania zhôd v dokumentoch, real-time indexovanie, dynamický clustering a import dát z SQL a NoSQL databáz. Taktiež ma bohatú podporu práce s dokumentami rôznych typov (Word, PDF a podobne) a mnoho ďalších funkcii. Medzi dôležitú funkcionalitu patrí možnosť využitia REST API na tvorbu dotazov nad údajmi a možnosť horizontálneho škálovania. [40]

Obr. 3.8: Logo projektu Apache Solr [40]

3.4 Relačná databázová technológia

Relačné databázové systémy sú využité počas vývoja a prevádzky informačných systémov za účelom ukladania dát centrálne, trvalo a v štrukturovanej forme. RDBMS sú integrované systémy pre konzistentný manažment tabuliek a ponúkajú funkcionalitu v podobe služieb a taktiež v podobe deskriptívneho jazyka SQL pre popis dát, ich výber a manipuláciu.

Každý RDBMS sa skladá z úložného priestoru a manažujúcej komponenty. Úložný priestor ukladá ako dáta, tak aj relácie medzi jednotlivými údajmi v tabuľkách. Okrem údajov rôznych užívateľských dát a aplikácii obsahuje taktiež preddefinované systémové tabuľky, ktoré sú potrebné pre databázové operácie. Tie obsahujú popisné informácie a môžu byť dotazované, avšak nie upravované užívateľom.

Najdôležitejšou časťou manažmentu systému je jazyk SQL, ktorým je možné dáta definovať, dotazovať a manipulovať (Data Definition and Manipulation Language (DDL/DML)), Táto komponenta systému taktiež obsahuje obslužné funkcie pre obnovu dát v prípade chyby, pre ochranu dát a ich zálohu. RDBMS je bežným základom BI systémov. [55]

PostgreSQL

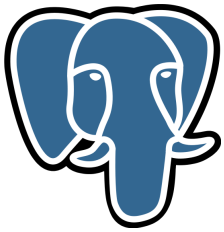
PostgreSQL je open-source objektovo-relačný databázový systém, ktorý používa a rozširuje SQL jazyk a to pridávaním rôznych vylepšení, ktoré bezpečne uložia a škálujú aj tie najkomplikovanejšie dátové úlohy[4]. Pôvod PostgreSQL siaha do roku 1986[5], kde začínal ako súčasť univerzitného projektu POSTGRES na Kalifornskej Univerzite. Dá sa teda hovoriť, že jeho aktívny vývoj prebiehal posledných 30 rokov.

PostgreSQL beží na všetkých hlavných platformách operačných systémov, pričom od roku 2001 oficiálne podporuje Atomicity, Consistency, Isolation, Durability (ACID) vlastnosti, vrátane rôznych iných vylepšení.

[4] Príkladom môže byť využitie PostgreSQL doplnku PostGIS v rámci geografického informačného systému, čo je špecifickým typom systému, ktorý musí vedieť pracovať s geografickými dátami celej planéty

[5] MySQL vznikol v roku 1995

PostgreSQL je preukázateľne vysoko-škálovateľný systém a to ako v oblasti samotného množstva dát, ktoré vie manažovať, tak aj v počte paralelných prístupov, ktoré dokáže obslúžiť. V produkcii aktuálne fungujú clustery, ktoré manažujú dáta vo veľkosti niekoľkých terabajtov a špecializované systémy, ktoré manažujú petabajty dát. [32]



Obr. 3.9: Logo projektu PostgreSQL [32]

Greenplum

Greenplum databáza je massively parallel processing (MPP) databázovým systémom s architektúrou, špeciálne navrhnutou na manažovanie veľkého objemu analytických dát a BI úloh.

MPP značí systémy, ktoré majú viac ako dva procesory a ktoré spolupracujú na vykonaní spoločnej operácie, pričom každý procesor má svoju vlastnú pamäť, operačný systém a disk. Greenplum využíva túto vysoko-výkonnú systémovú architektúru na distribúciu záťaže medzi viacero n-terabajtových dátových skladov a môže využiť všetky systémové zdroje v rámci paralelného spracovania údajov.

Greenplum databáza je založená na technológii PostgreSQL. Je to vo svojej podstate niekoľko PostgreSQL diskovo-orientovaných databázových inštancii, ktoré jednajú spoločne ako jeden súdržný Database Management System (RDBMS). Greenplum vychádza z verzie PostgreSQL 8.3.23 a je vo väčšine ohľadov veľmi podobný PostgreSQL. Hlavne z ohľadom na podporu jazyka SQL, vylepšení, možnosti konfigurácie a funkcionality pre koncových užívateľov. Databázový užívatelia komunikujú s databázou Greenplum rovnako, ako by komunikovali s bežným PostgreSQL RDBMS[6].

Greenplum databáza môže byť využitá v rámci append-optimized (AO) úložného formátu pre hromadné nahratie a čítanie dát, pričom poskytuje výkonnostnú výhodu nad haldovými formátom tabuliek[7]. AO úložisko poskytuje kontrolné súčty (*checksum*) pre ochranu dát a kompresiu na úrovni

[6] Ako príklad môže by spomenuté, že v analytickej časti bola záloha PostgreSQL použitá na naplnenie Greenplum databázy (s miernymi úpravami v oblasti vytvárania tabuliek)

[7] základný typ tabuľky pre PostgreSQL

stĺpcov. Ako stĺpcovo orientované, tak aj riadkovo orientované AO tabuľky môžu byť komprimované. [19]

Obr. 3.10: Logo projektu Greenplum [39]

Zed store

Zed store je open-source kompresná stĺpcová vrstva pre ukladanie dát v PostgreSQL, ktorá je vyvíjaná s cieľom integrovania do jadra PostgreSQL časom. Stĺpcovo orientovaná technológia je často preberaná v rámci PostgreSQL komunity a je zároveň populárnou žiadosťou od firemných klientov. To bolo hlavnou motiváciou za uvedením API rozhrania k tabuľkám v rámci jadra PostgreSQL na začiatku roku 2019. API, ako meno napovedá, poskytuje možnosti písania vlastných doplnkov pre ukladaciu vrstvu. Zed store je striktnou implantáciou tohto API. [42]

Zed store vznikol začiatkom roka 2019 a to po uvedení API rozhrania nad tabuľkami v PostgreSQL. Tým sa skladá z dlhoročných prispievateľov ako do projektu PostgreSQL, tak aj do projektu Greenplum. [42]

Zároveň je Zed store predpokladaným myšlienkovým nasledovníkom [52] projektu Greenplum, ktorý by malo byť možné využiť ako plnohodnotnú náhradu za danú technológiu s jediným rozdielom a to že každý PostgreSQL užívateľ by mohol použiť stĺpcovo orientovanú technológiu. Nakoľko však Greenplum implementácia úložiska predchádza API rozhraniu v PostgreSQL, táto implementácia trpí neustálymi problémami v rámci zlučovania zmien s projektom PostgreSQL. To obmedzuje počet vylepšení, ktoré je možné do projektu Greenplum implementovať bez prípadnej re-implementácie z dôvodov zmien v PostgreSQL architektúre v priebehu času. [42]

Aktuálne projekt nie je v stave, ktorý by umožňoval jeho produkčné nasadenie, avšak je to zaujímavá technológia, ktorej vývoj je vhodné sledovať do budúcna. [52]

3.5 Key-value databazová NoSQL technológia

Najjednoduchším spôsobom ako uložiť dáta, je priradiť každej premennej hodnotu alebo kľúč. Na úrovni hardvéru napríklad procesor pracuje s registrami, ktoré sú založené na rovnakom princípe. Programovacie jazyky zase používajú rovnaký princíp pri previazaných poliach (*associated arrays*). Rovnako najjednoduchší model, ktorý je možné použiť na úrovni databázového systému je ten, ktorý ukladá dátové objekty ako hodnoty, ktoré sú použité ako kľúč pre iné dátové objekty.

Kľúče môžu byť štrukturované iba pomocou špeciálnych znakov, akými sú dvojbodka a lomítko. To umožňuje definovať menný priestor (*namespace*), ktorý môže reprezentovať základnú dátovú štruktúru. Okrem toho key-value úložný priestor nepodporuje akýkoľvek iný typ štruktúry a to vrátane zanorenia[8] alebo referencii (`FOREIGN KEY`).

[8] Toho je však možné dosiahnuť o vrstvu vyššie, ako je poukázané v popise technológie MongoDB

Key-value úložisko je bezschémové, čo znamená, že dátové objekty môžu byť uložené v ktorýkoľvek čas v ľubovolnom formáte a to bez potreby akýchkoľvek metadát, akými sú tabuľky alebo stĺpce, ktoré by museli byť definované dopredu. Ísť bezschémovou cestou a bez referenčnej integrity dát robí key-value úložisko výkonné pre dotazy, jednoduché na delenie a flexibilné bez ohľadu na dátový typ informácii, ktoré majú byť uložené. [55]

Databáza je typu key-value v prípade, že má nasledujúce vlastnosti [55]:

- Existuje rada, ktorá identifikuje dátové objekty (kľúč)
- Pre každý kľúč existuje presne jeden asociovaný dátový objekt
- Špecifikovanie kľúča umožňuje dotazovanie, ktoré je asociované s hodnotou v databáze

MyRocks

MyRocks je open-source úložný engine, ktorý bol pôvodne vyvíjaný spoločnosťou Facebook. Jedná sa o MySQL úložný engine, ktorý je integrovaný s technológiou RocksDB. Technológia poskytuje vylepšené úložisko pre SSD technológiu s výkonom, zameraným hlavne na efektívnejšie čítanie a zápis uložených dát.

Technológia typicky dáva lepší výkon pre webovo škálovateľné aplikácie. Môže byť teda ideálnym úložiskom pre riešenia, pri ktorých je vyžadovaná lepšia kompresia a efektivita input-output (IO) operácii, nakoľko technológia má desať násobne menšie množstvo zápisov v porovnaní s enginom InnoDB[9], čo umožňuje predĺženie obmedzenej životnosti SSD diskov a zlepšuje celkovú priepustnosť systému.

Zároveň má technológia MyRocks až dvojnásobne lepší kompresný pomer v porovnaní s komprimovaným InnoDB a tri až štyrikrát lepšiu kompresiu dát v porovnaní s nekomprimovaným InnoDB, čo znamená, že vo výsledku je zabraných menej priestoru na disku.

Okrem podpory MySQL má technológia taktiež podporu v MariaDB. Medzi vážnu nevýhodu však patrí chýbajúca podpora cudzích kľúčov, čo je pravdepodobne spojené s obmedzujúcim faktorom na úrovni key-value technológie. [27] [4]

[9] Použitý defaultne v MySQL

RocksDB

RocksDB je log-structured merge-tree (LSM) databáza optimalizovaná pre SSD disky. Používa menej miesta, zapisuje menší objem dát a zároveň číta menšie množstvo údajov v porovnaní s B-tree algoritmom, ktorý je defaultne implementovaný napríklad v MySQL, MariaDB alebo PostgreSQL technológii. Jedná sa o základnú technológiu, nad ktorou je postavený engine MyRocks. [54]

Obr. 3.11: Logo reprezentujúce RocksDB engine použitý v MyRocks [54]

3.6 Dokument orientovaná NoSQL databázová technológia

Dokument orientovaná databázová technológia kombinuje absenciu schémy s možnosťou štruktúrovania uložených dát. Aj keď meno technológie to môže implikovať, dokument orientovaná technológia neukladá dokumenty ako webové stránky, video alebo audio dáta, ale štruktúrované dáta v podobe záznamov, ktoré sa nazývajú *dokumenty*. Dokument ukladá informácie, ktoré boli vytvorené špecificky za účelom využitia vo webových technológiách. Môžu byť teda jednoducho integrované vo webových technológiách, akými je napríklad JavaScript alebo HTTP.

Okrem toho je technológia pripravená na horizontálne škálovanie a to kombináciou viacerých počítačových systémov do jedného integrovaného systému, ktorý rozdistribuuje dáta technológiou, ktorá sa volá *sharding*.

Hlavným zameraním je spracovanie veľkého objemu heterogénnych dát, pričom v prípade webových dát napríklad zo sociálnych sieti alebo vyhľadávacích enginov a novinových portálov nie je neustála konzistencia dát až tak nutná. Bezpečnostne citlivé služby, akými je napríklad online bankovníctvo, ktoré sa silne spoliehajú na obmedzenia na úrovní schém a garantovanú konzistenciu uložených údajov, sú väčšinou výnimkou.

Dokument orientovaná technológia je úplne bezschémová, čo znamená, že nie je treba definovať schémy pred samotným vkladaním údajov. Zodpovednosť za schémy je teda presunutá na užívateľa alebo spracovateľskú aplikáciu.

Nevýhoda tohto prístupu spočíva v chýbajúcej referenčnej integrite a normalizácii, avšak absencia schémy umožňuje extrémnu flexibilitu v ukladaní širokého množstva údajov, ktoré sú tak žiadané pri spracovaní Big data. [55]

Dokument orientovaná technológia má nasledujúce vlastnosti [55].

- Key-value technológia
- Dátové objekty sú uložené ako hodnoty pre kľúče zvané dokumenty
- Kľúče sú použité pre identifikáciu
- Dokumenty obsahujú dátovú štruktúru vo forme rekurzívne vnorených dvojíc key-value, ktoré sú bez referenčnej integrity
- Dátová štruktúra je bezschémová, čo znamená, že ľubovolné atribúty môžu byť použité v ľubovolnom dokumente bez deklarovania schémy na začiatku

MongoDB

MongoDB je univerzálnym, dokument orientovaným distribuovaným databázovým systémom, ktorý je vybudovaný primárne pre moderné webové aplikácie v ére cloudu. Technológia ukladá dáta vo flexibilných, JSON-u podobných dokumentoch, čo znamená, že každý záznam môže byť odlišný a každá dátová štruktúra môže by zmenená v priebehu času.

Dokumentový model mapuje objekty v aplikačnom kóde, čo robí prácu s dátami jednoduchú. Priame (*ad-hoc*) dotazovanie, indexovanie a agregácia v reálnom čase poskytujú efektívny prístup a analýzu dát.

MongoDB bol už pri samotnom návrhu plánovaný ako distribuovaný databázový systém, čo znamená, že vysoká dostupnosť, horizontálne škálovanie a geografická distribúcia dát je zabudovaná a jednoducho použiteľná. [26]

Obr. 3.12: Logo technológie MongoDB [26]

3.7 Stĺpcovo orientovaná databázová NoSQL technológia

Aj napriek tomu, že key-value technológia je schopná spracovať veľké objemy dát výkonne, štruktúra je však primitívna a často je potrebné dátovú maticu štruktúrovať dodatočne pomocou schémy.

Stĺpcovo orientovaná trieda (*column-oriented family*) vylepšuje koncept key-value tým, že pridáva dodatočne schému. V praxi sa totiž ukázalo, že použitie schémy je efektívnejšie vtedy, keď príde k optimalizovaniu operácii čítania v relačných databázach a to pri použití stĺpcovo orientovaného úložiska namiesto riadkovo orientovaného.

Dôvodom je fakt, že všetky stĺpce v jednom riadku sú málokedy potrebné naraz, ale existujú skupiny stĺpcov, ktoré sú často čítane spoločne. Preto za účelom optimalizovania prístupu je vhodné štruktúrovať dáta do takých skupín, v akých sa budú čítať.

Túto úlohu spĺňa tzv. trieda stĺpcovo (column-family) orientovaných databázových systémov, ktoré sú pomenované po tejto metóde. Dáta sa ukladajú nie v relačných tabuľkách, ale vo vylepšených a štrukturovaných multidimenzionálnych kľúčových priestoroch.

Medzi najväčších priekupníkov tejto technológie patrí Google s jeho cloudovým riešením v podobe Bigtable[10] pre distribuované úložisko štruktúrovaných dát. [55]
Vlastnosti stĺpcovo orientovaných NoSQL databázových systémov sú nasledujúce [55]:

- Dáta sú uložené v multidimenzionálnych tabuľkách
- Dátové objekty sú adresované prostredníctvom riadku kľúčov
- Vlastnosti objektov sú adresované prostredníctvom stĺpcu kľúčov
- Stĺpce tabuľky sú zoskupené do tzv. triednych stĺpcov (column-family)
- Schéma tabuľky môže referencovať jedine na triedny stĺpec. Pre každý triedny stĺpec môže byť použitý ľubovolný počet stĺpcov s kľúčom
- V distribuovanej, rozfragmentovanej architektúre sú dáta z rovnakého triedneho stĺpcu preferované v rámci jedného fyzického úložiska a to za účelom optimalizovania času odozvy

[10] `https://cloud.google.com/bigtable`

Apache Cassandra

Cassandra je NoSQL distribuovaná databáza spadajúca do triedy stĺpcovo orientovaných databáz. Z povahy samotného konceptu NoSQL je ľahká (*lean*)[11], open-source, nerelačná a distribuovaná. Medzi jej silné stránky patrí horizontálne škálovanie, distribuovaná architektúra a flexibilný prístup k definícii schémy.

NoSQL databázy ponúkajú rýchlu, ad-hoc organizáciu a analýzu extrémne veľkého objemu nesúrodých dátových typov. O to viac je to dôležité v posledných rokoch s nárastom využitia Big data a potreby rýchleho škálovania databázového systému v cloude. Cassandra patrí práve medzi NoSQL databázy, ktoré venovali čas adresovaniu limitujúcich faktorov, predchádzajúcich manažment technológii akými sú napríklad klasické SQL databázy. [9]

[11] Myslené z pohľadu systémových zdrojov na samotný chod

Kapitola 4

Analýza súčasného stavu

V nasledujúcej kapitole bude popísaný aktuálny stav. Bude popísané, prečo je vhodné a dalo by sa povedať až potrebné, zbierať uniknuté údaje. Následne sú analyzované existujúce riešenia, ktoré sú verejne dostupné a budú popísané ich silné i slabé stránky. V poslednej časti kapitoly sú prebraté možnosti, ako dáta spracovať a popíšu sa úskalia a problémy, ktoré sa objavili a objavia pri ich spracovaní.

4.1 Prečo zbierať údaje

V oblasti počítačovej bezpečnosti sa hovorí že:

> Neexistuje niečo ako dátový únik. Existuje iba nečakaná externá záloha systému.

Tento výrok dokonale vystihuje potrebu analýzy a spracovania uniknutých údajov. Je v záujme štátu ale aj súkromných spoločnosti mať prehľad o dostupných informáciach, ktoré sa zdieľajú a s ktorými sa obchoduje v hackerských skupinách. Podľa výsledkov analýzy spoločnosti Pandas security [2] až 52% užívateľov využíva rovnaké prihlasovacie údaje na viacerých službách. To je niečo, čomu sa napríklad správnym nastavením politik hesiel zabrániť nedá. Zároveň sa tým otvárajú dvere pre útočníkov. Ako ukazuje report spoločnosti F5 [1], počet únikov rastie a toho využívajú hlavne sofistikovanejšie hackerské skupiny.

Dodatočné uniknuté údaje, akými môže byť adresa zamestnancov spoločnosti, ich osobné záujmy a koníčky je možné využiť pri spear phishing útoku.

Vzniká teda potreba tieto údaje zbierať, aby sa Security Operation Center (SOC) vedelo dostatočne zabezpečiť pred prípadným útokom a upozorniť svojich zamestnancov o ich odcudzených

údajoch. Príkladom môže byť únik [20] prihlasovacích údajov do podnikových VPN v roku 2020, ktoré následne boli zneužité na ransomware útoky. V prípade včasného získania údajov národným CSIRT tímom, škodám, ktoré boli pravdepodobne v rádoch miliónov, by sa dalo z veľkej časti zabrániť.

Zároveň sú zozbierané údaje skvelým zdrojom, ako pre výskumné účely v oblasti bezpečnosti, tak aj pre prípadne *wargames* (simulácia útoku a obrany) v rámci organizácie. Tieto údaje robia simulovaný útok na spoločnosť reálnejším, nakoľko je možné vytvoriť scenár na základe informácii, ktoré by mali útočníci bežne k dispozícii.

V prípade úspešného boja voči kybernetickým hrozbám je znalosť možností, ktoré má protistrana k dispozícií, kritická. Hĺbková znalosť a pochopenie údajov je pritom dôležité taktiež pre správne manažérske rozhodovanie v oblasti bezpečnosti [61].

4.2 Existujúce riešenia

Myšlienka analýzy, zbierania a spracovania uniknutých databáz nie je nová. Existuje niekoľko rôznych riešení, pričom každé ma svoje klady a zápory.

haveibeenpwned.com

Webová služba `haveibeenpwned.com` patrí k jednej z najznámejších webových služieb, ktoré umožňujú skontrolovať svoju emailovú adresu alebo login voči databáze uniknutých údajov. Za jej tvorbou stojí austrálsky bezpečnostný konzultant Troy Hunt. Stránka aktuálne poskytuje databázu prihlasovacích údajov z 530 webových stránok a 114 102 listov (*pastes*). Celkovo databáza obsahuje viac ako 11,2 miliardy účtov.

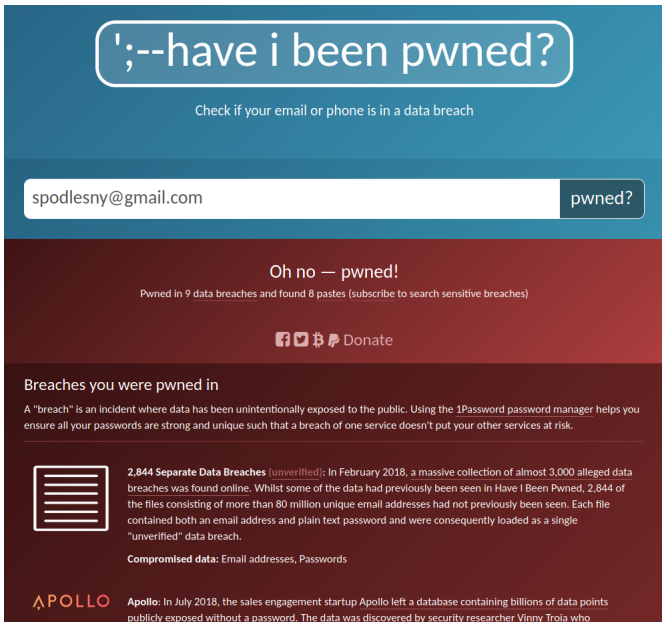
Ako je možné vidieť na obrázku 4.1, stránka nezobrazí samotné prístupové údaje, iba informácie o tom, v ktorých zdrojoch sa daná adresa alebo login nachádza a aké údaje boli odcudzené.

Po technickej stránke služba využíva služby Azure cloud a špecificky [43]:

- SQL Azure database
- Azure Table Storage

SQL Azure Storage je cloudová verzia klasického RDBMS systému, zatiaľ čo Azure Table Storage je NoSQL databázov pravdepodobne patriacej do kategórie stĺpcovo orientovanej NoSQL databázovej technológie. K tomuto záveru sa došlo na základe toho, že služba vyžaduje schému dát.

Klasická relačná databáza sa využíva prevažne na webové účely a obsahuje informácie ako zoznam leakov, databázu odberateľov notifikácii a podobne. Azure Table Storage je pre zmenu využitý na uloženie obsahu samotných leakov a na vyhľadávanie informácii vo vnútri leakov.



Obr. 4.1: Ukážka webovej stránky `haveibeenpwned.com` [58]

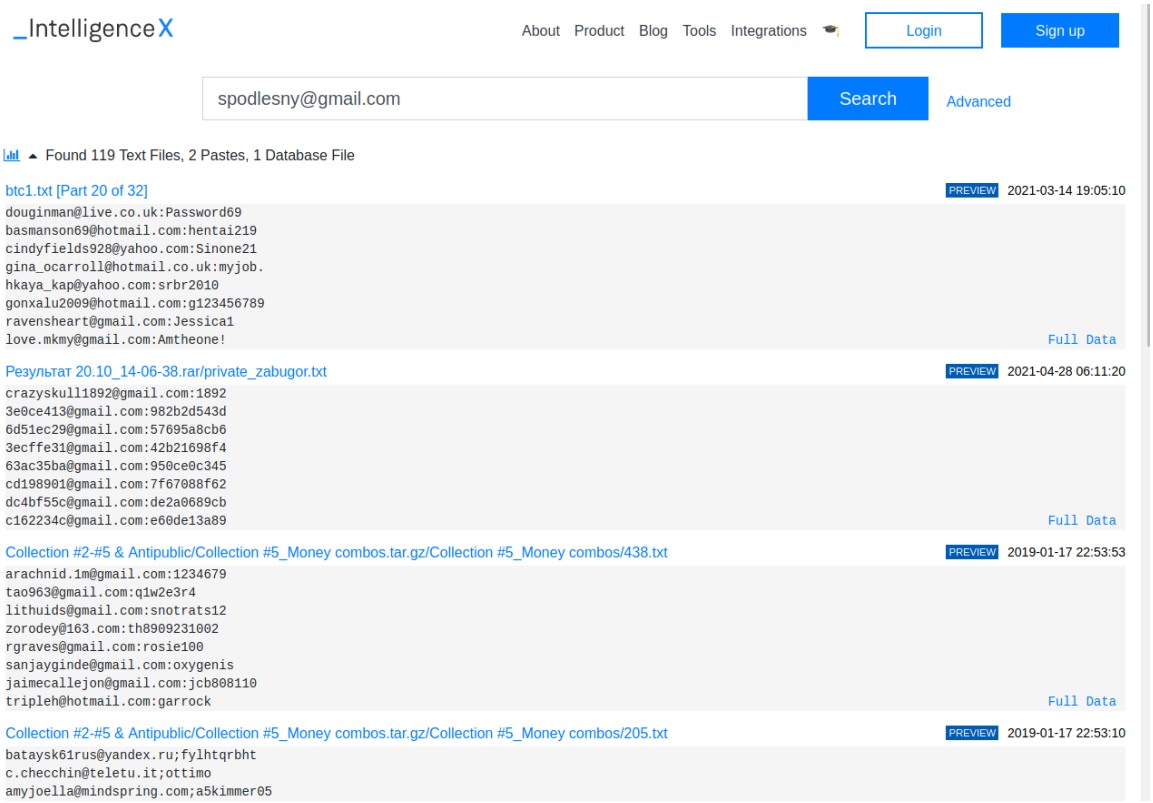
Intelligence X

Intelligence X je nezávislou Európskou technologickou spoločnosťou, ktorá vznikla v roku 2018. Spoločnosť ma sídlo v Prahe.

Misiou spoločnosti je vyvinutie a udržiavanie vyhľadávacieho enginu a archivácia dát. Intelligence X sa odlišuje od ostatných vyhľadávacích enginov nasledujúco:

- Vyhľadávanie funguje na základe tzv. selektorov, ktoré umožňujú hľadať špecifický typ údajov (email, IP adresu, URL, Bitcoin adresu, hash a podobne)
- Vyhľadávač hľadá na miestach, akým je napríklad darknet, platformy na zdieľanie dokumentov, WHOIS, verejne dostupné uniknuté dáta a iné
- Zachováva historické dáta výsledkov, podobne ako funguje Wayback Machine v projekte archive.org

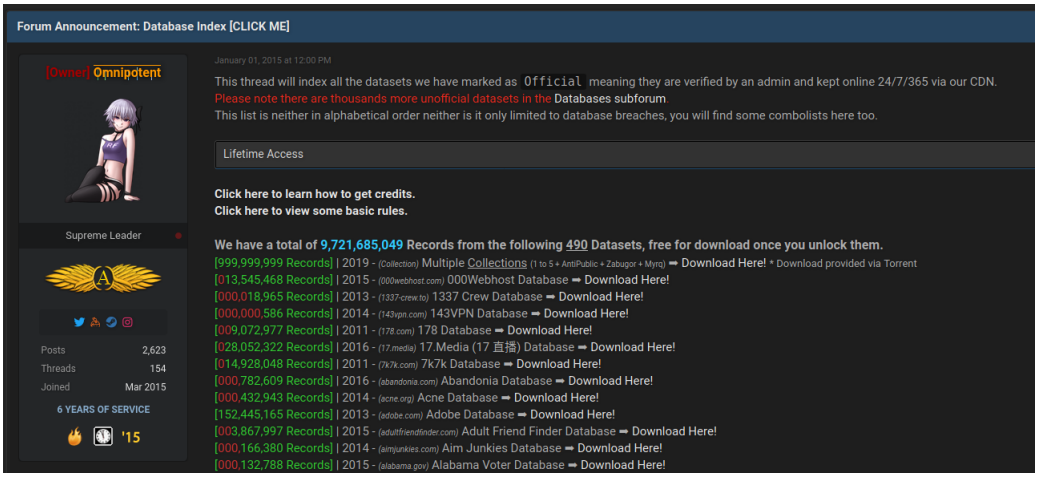
Služba aktuálne prezentuje, že má viac ako 40,6 miliardy záznamov. [21] Toto číslo však nie je možné overiť. Podobne ako nie je možné overiť rýchlosť pridávania nových záznamov. Na obrázku 4.2 je zobrazené základné vyhľadávanie na webovej stránke.



Obr. 4.2: Záber z webového rozhrania Intelligence X [58]

4.3 Formát a veľkosť dát

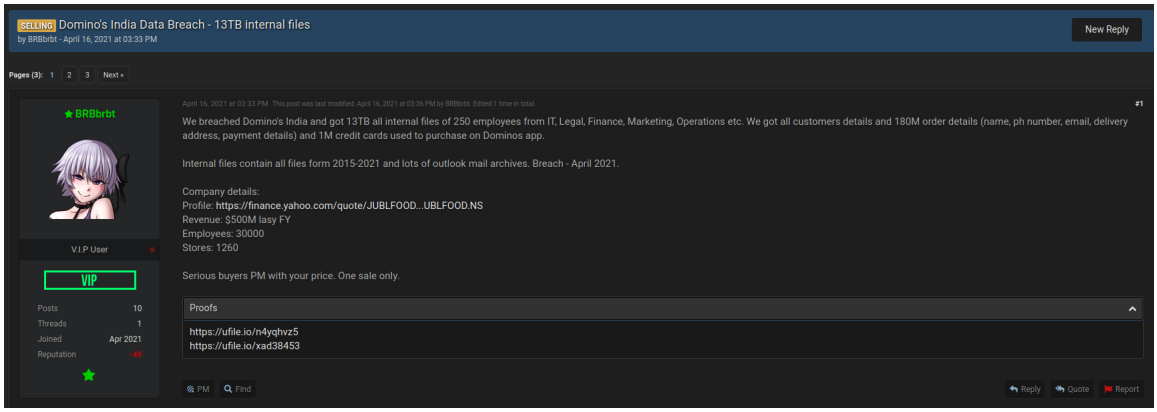
Každý rok uniknú milióny prihlasovacích údajov a osobných údajov o zákazníkoch. Spomenúť rok 2021, objavili sa medzi uniknutými údajmi informácie o zákazníkoch zo sociálnej siete LinkedIn [36] a Facebook [15]. Tieto informácie sa často zdieľajú na rôznych undreground hackerských fórach a kanáloch. Ako je možné vidieť na obrázku 4.3, počet dostupných údajov sa pohybuje v miliardách.



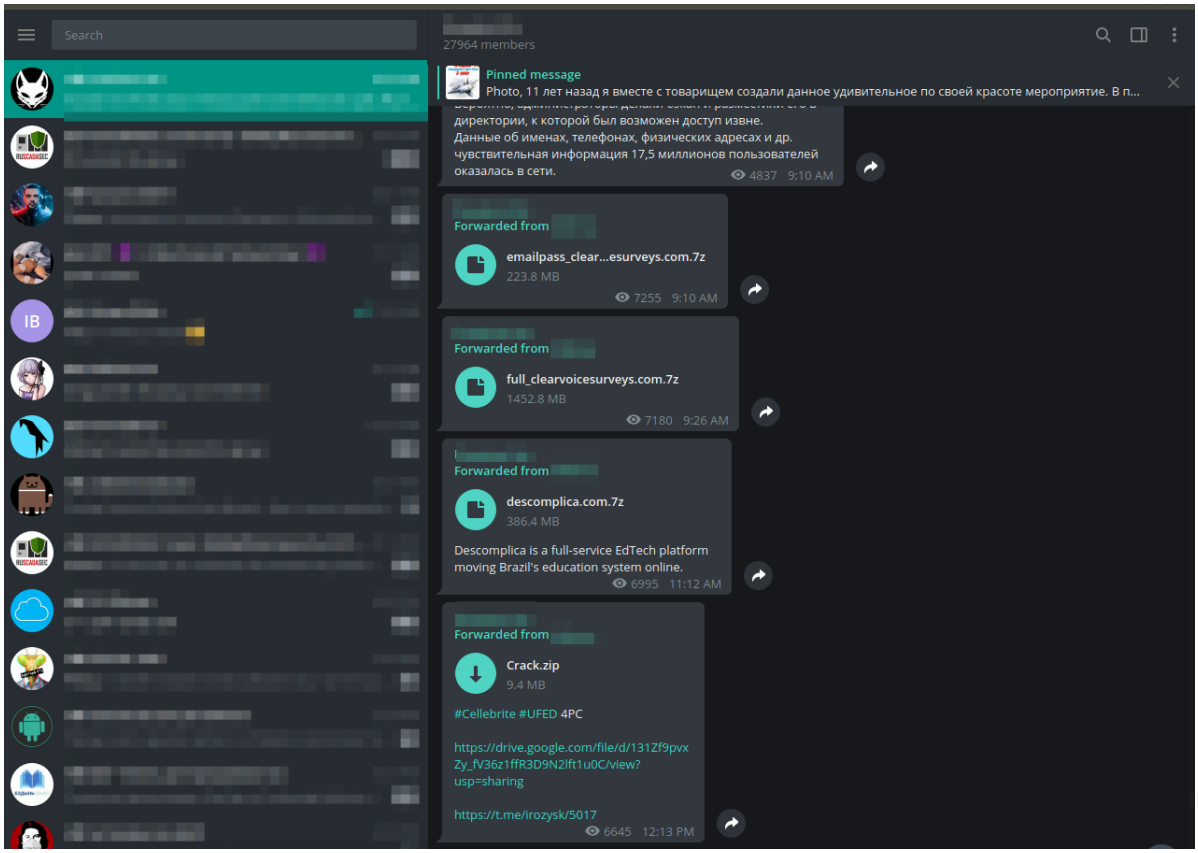
Obr. 4.3: Záber jedného z najznámejších underground obchodov s uniknutými údajmi [58]

Pritom tento počet obnáša údaje len z overených únikov a väčšinou sa jedná o údaje v podobe prihlasovacieho mena a emailu. Taktiež ako je možné vidieť na obrázku 4.5, ktorý pochádza z ruskej skupiny v aplikácii Telegram, na internete kolujú rôzne ďalšie databázové úniky, ktoré sú dostupné, ale neoverené. Toto býva často problém, ako ukázala kauza indického MobiKwik, ktorý podľa svojho oznámenia [24] "nenašiel žiadne dôkazy o úniku", pritom ako je možné vidieť na obrázku 4.4, údaje daného poskytovateľa mobilných platieb v Indii sa predávajú na underground fórach.

Podobnú kauzu nedávno zažila aj relatívne nová sociálna sieť Clubhouse, keď sa na internete objavila databáza užívateľských informácii z danej sociálnej siete. Databáza obsahovala 1,3 milióna prihlasovacích údajov a to vo formáte SQLite databázy. Podľa vyjadrenia spoločnosti sa však nejednalo o prekonanie zabezpečenia spoločnosti, ale o zneužitie API rozhrania na získanie daných informácii, ktoré sú však verejne dostupné v rámci sociálnej siete [10].



Obr. 4.4: Ponuka uniknutých údajov poskytovateľa platieb MobiKwik [58]



Obr. 4.5: Záber na ruský telegram kanál, ktorý zdieľa uniknuté databázy [58]

Problém spracovania dát sa skladá z troch hlavných častí, ktoré sú veľmi podobné generickým problémom spracovania Big data, na ktoré bolo poukázané v sekcii (3.1):

- Konzistencia dát
- Formát dát
- Veľkosť dát

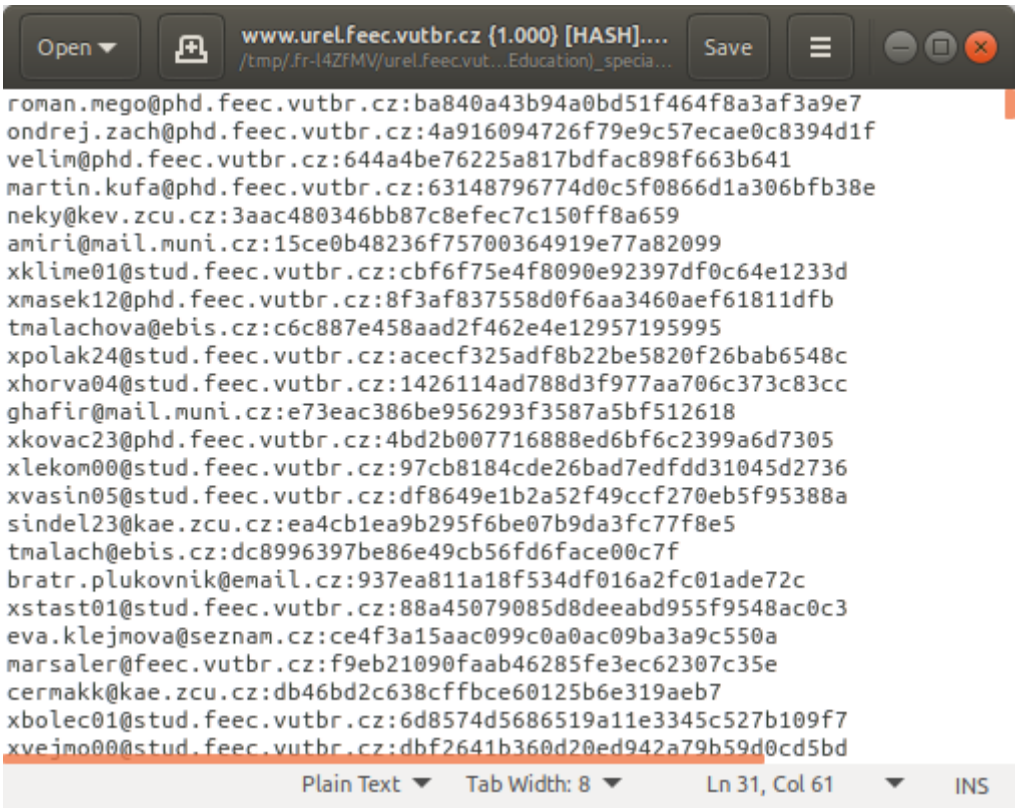
Zároveň sa objavuje špecifická kombinácia problémov a to:

- Veľké množstvo dát, ku ktorým by mal byť možný ideálne real time prístup
- Rôzne druhy formátov
- Rôzne hodnoty na vstupe
- Rôzna kvalita zdrojov

Konzistencia dát

Veľké množstvo dostupných dát je upravených a obmedzených na formát podobný CSV, kde je čiarkou, bodkočiarkou alebo medzerou oddelené prihlasovacie meno, prípadne email a heslo alebo jeho hash (viď 4.6). Účelovo alebo z dôvodu chybného automatizovaného spracovania tomu však nie je vždy. Časté sú rôzne textové chyby. Medzi najčastejšie patrí:

- Použitie null znaku (\x00) namiesto medzery
- Použitie prázdneho riadku v dátovej časti
- Porušenie formátu dátového súboru (napríklad vynechaním hesla)
- Zámenou oddelovacích znakov ;,: v rámci jedného súboru
- Miešaním loginov a emailových adries



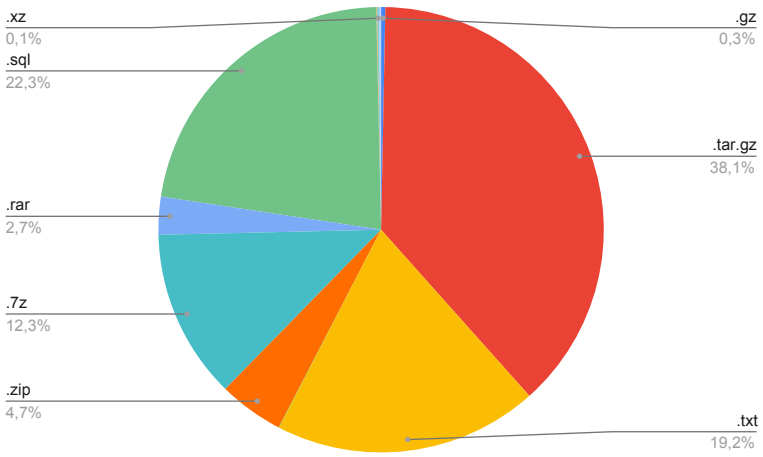
Obr. 4.6: Náhľad na uniknutú databázu webovej stránky `urel.feec.vutbr.cz` [58]

Celková veľkosť uložených dát

V prípade celkovej veľkosti uložených dát treba počítať s nasledujúcimi hlavnými parametrami:

- Priestor zaberajúci na disku
- Celková veľkosť spracovaných dát (t.j. po rozbalení)
- Priestor zaberajúci po spracovaní
- Najväčšia veľkosť archívu po rozbalení

Priestor zaberajúci na disku je potrebné vedieť pre nastavenie veľkosti systému, na ktorom budú uložené originály súborov, s ktorými sa bude pracovať. Aktuálne má spoločnosť k dispozícii viac ako 2TB dát (2,612TB), ktoré nazbierala v priebehu času. Očakáva sa, že po spustení projektu množstvo údajov narastie na niekoľko desiatok terabajtov.

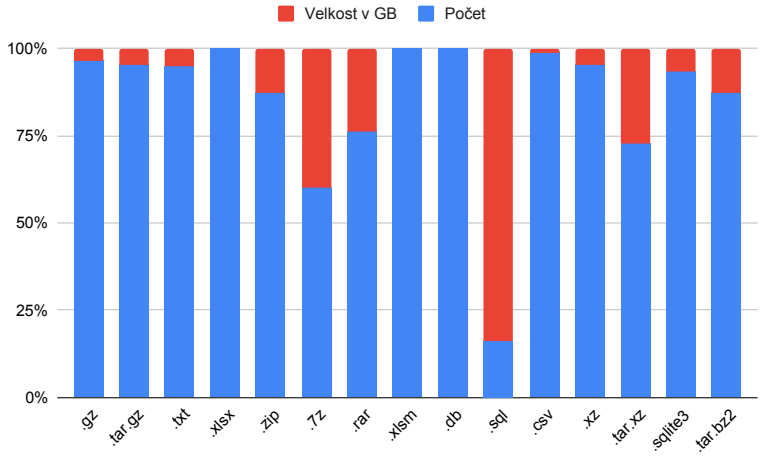


Obr. 4.7: Zloženie veľkosti súborov na základe prípony [58]

Pritom ako je možné vidieť na obrázku 4.7, viac ako tretina údajov je v komprimovanej podobe a po rozbalení by veľkosť mohla niekoľkonásobne narásť, nakoľko úroveň kompresie textového súboru môže dosahovať až 80% a vypočítná priemerná hodnota je 53%[1]. Zároveň sa ukázalo, že existuje určitý nepomer medzi veľkosťou súborov a počtom súborov, kde do značnej miery vyčnieva hlavne formát `.sql`, čo je textová forma zálohy databázy. Tento fakt je možné vidieť na obrázku 4.8. Celkovo je medzi dátami menej ako jedno percento SQL súborov, pritom tvoria 22% veľkosti celkového objemu dát. **Jedno z odporúčaní pri finálnom návrhu je robiť pravidelnú**

[1] Tento odhad je však treba brať s rezervou, nakoľko údaje o kompresii sú získane príkazom `gzip -l`, ktorý však nie je presný

analýzu súborov a v prípade formátov, akým je `.sql`, ich pred uložením dodatočne komprimovať.



Obr. 4.8: Pomer veľkostí súborov k ich celkovému počtu [58]

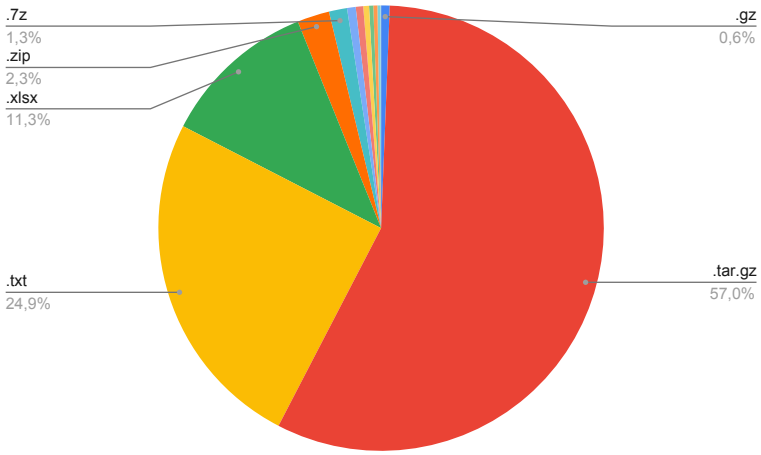
Obsah dát a ich formátovanie

Pre účely návrhu a implementácie systému na analyzovanie a spracovanie uniknutých údajov sú síce využité iba verejne dostupné wordlisty, známe pod názvom `Collection 1` a `City0Day`, ktoré sú vo formáte podobnom CSV. Pre implementáciu a nasadenie výslednej architektúry je ale potrebné počítať aj s inými formátmi súborov. Analýzou dostupných dát boli zistené nasledujúce počty formátov:

- 180 .rar
- 866 .zip
- 21075 .tar
- 92 .xz
- 18 .gz
- 4 .sqlite3
- 98 .sql
- 4 186 .xlsx
- 200 xlsm
- 58 .csv
- 516 .7z
- 183 .db
- 10 051 .txt

K štatistike typu súborov je však potrebné pristupovať obozretne. Ako je možné vidieť na obrázku 4.9, viac ako polovica dát je v archíve a ten je len obalom pre niektorý z textových alebo binárnych formátov.

Pre presnejšie štatistiky by bolo potrebné rozbaliť všetkých viac ako dvadsaťdva tisíc archívov a nakoľko podľa sekcie 4.3 sa jedná o viac ako jeden terabajt údajov v komprimovanej podobe, aktuálne chýbajú zdroje na úspešné uloženie tak veľkého množstva dát. Je však možné počítať so vzorkou, ktorá je vytvorená z tohto veľkého počtu archívov. Pre naše účely to bude `Collection 1` a `City0Day`, ktoré spoločne reprezentujú 98% archívov s príponou `tar.gz`, avšak len necelé 2% celkového objemu dát.



Obr. 4.9: Percentuálne zobrazenie počtu súborov na základe prípony [58]

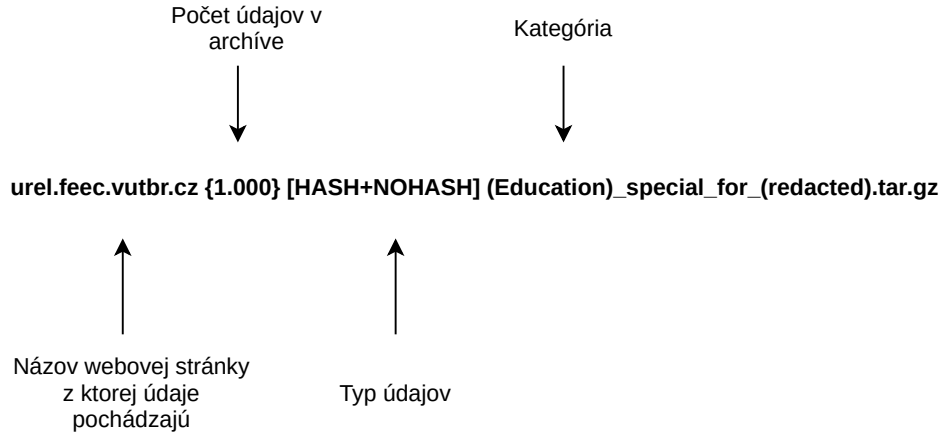
Štruktúra City0Day archívu

Archív `City0Day` sa objavil na Internete v novembri roku 2020 a skladá sa z celkovo 20 650 `tar.gz` súborov, čo tvorí 98% všetkých súborov s príponou `tar.gz`.

Štruktúra dát je vďaka tomu, že všetky údaje pochádzajú z jedného zdroja, veľmi podobná pre väčšinu súborov:

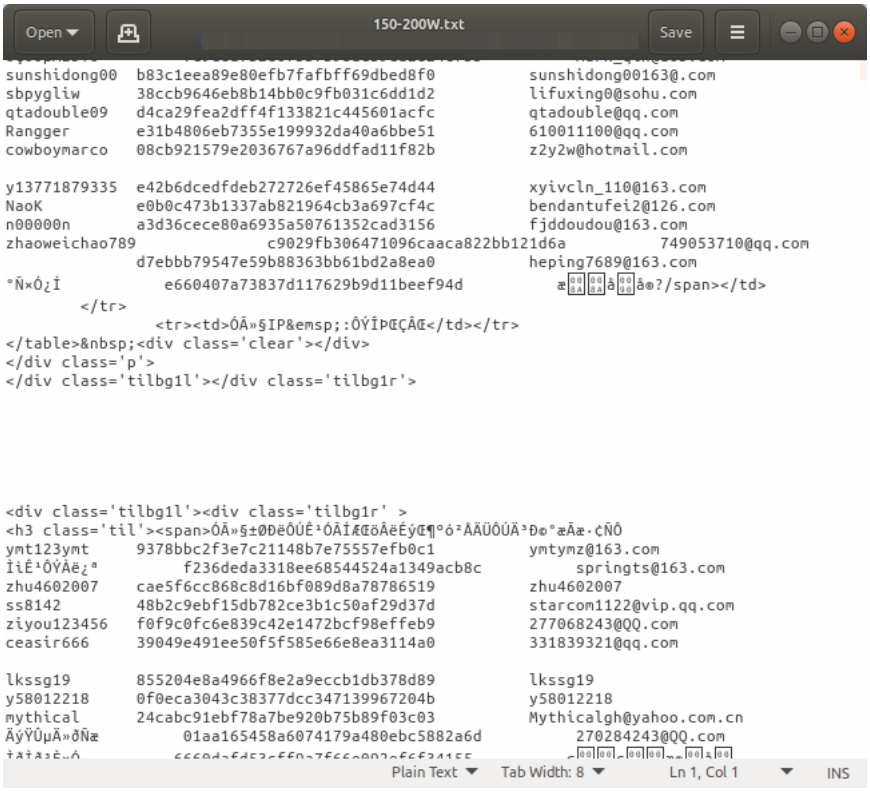
- Originálny súbor s loginom a hashom
- Súbor obsahujúci dešiforvané heslá a loginy (HASH+NOHASH)
- Súbor obsahujúci nedešifrované heslá a loginy (HASH)
- Súbor obsahujúci heslá, ktoré neboli šiforvané (NOHASH)

Formát pomenovania archívu má v aktuálnom prípade nasledujúci formát:

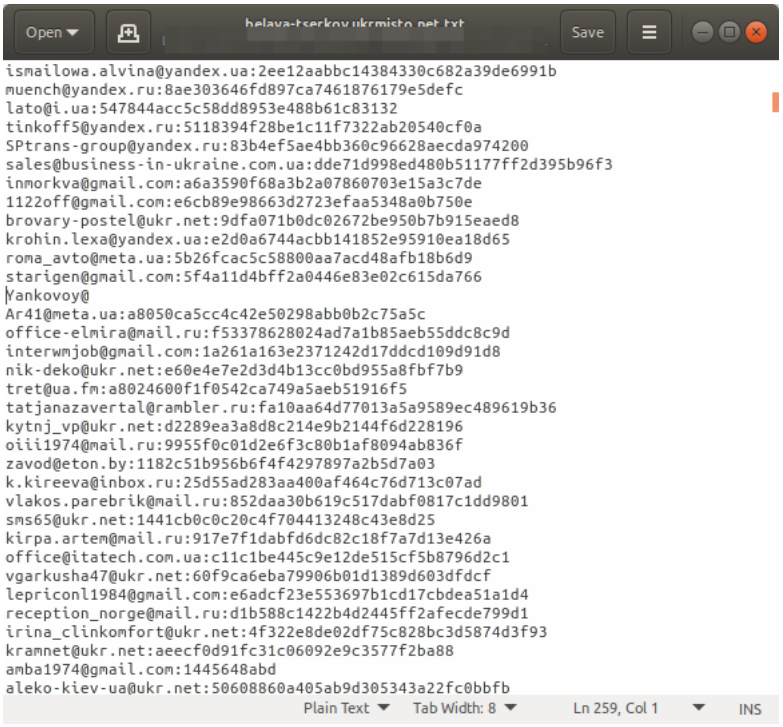


Obr. 4.10: Formát City0Day archívu [58]

Celkovo je v 20 650 archívoch obsiahnutých viac ako 540 994 590 záznamov. Archív sa zaraďuje do kategórie tzv. wordlistov, nakoľko je dopredu formátovaný a neobsahuje iné informácie ako prihlasovacie meno a heslo (prípadne hash hesla). V prípade textových súborov je situácia podobná, pričom veľká časť dát je v podobe wordlistov. Aj napriek tomu sa môžu často objaviť chyby z formátovania ako je zobrazené na obrázkoch 4.11 a 4.12.



Obr. 4.11: Ukážka wordlistu narušená kódom HTML [58]



Obr. 4.12: Ukážka chyby, pri ktorej je porušený formát súboru [58]

Štruktúra Collection 1

Archív `Collection 1` sa objavil na Internete začiatkom roku 2019 [3] a obsahuje viac ako 2 miliardy záznamov. Obsah je zložený z rôznych zdrojov a dáta obsahujú veľké množstvo duplicít.

Údaje sú poskytnuté v komprimovanej podobe formátu `tar.gz` a to celkovo v 30 archívoch, pričom najväčší má 10,9 GB a najmenší 6,7 MB. Obsah archívu je rôznorodý. Najčastejšie obsahuje desiatky, až stovky textových súborov, pričom mnohé sú pomenované numericky (`1.txt`, `2.txt`, `3.txt`), avšak nájdu sa aj archívy ako `Collection #1_OLD CLOUD_Hacking combos.tar.gz`, ktorý je možné vidieť vo výpise 1 a ktoré v názve[2] majú cyriliku. Zároveň nie sú súbory pomenované numericky, ale podľa stránok, z ktorých údaje pochádzajú. Pre zachovanie maximálneho množstva informácii je preto potrebné vo výslednej architektúre zaznamenať aj presný názov, z ktorého súbor pochádza.

```
Расшифровка D4tabase.com (Forum) [21k].txt
Расшифровка DemonForums.net [6k].txt
Расшифровка HackForums.net [120k].txt
Расшифровка BlackHatWorld.com [70k].txt
Расшифровка DashHacks.com [149k].txt
Расшифровка FreeHack.com [25k].txt
Расшифровка RootKit.com [63k].txt
```

Listing 1: Obsah archívu `Collection #1_OLD CLOUD_Hacking combos.tar.gz`, so špecifickým formátom [58]

Zaujímavý je taktiež archív `Collection #1_EU combos_1.tar.gz`, ktorý sa na prvý pohľad správa valídne, avšak jeho rozbaľovanie skončí chybou, nakoľko je pravdepodobne pokazený. **Výsledné riešenie by teda malo vedieť podobné archívy identifikovať, upozorniť na ne a pokračovať ďalej v spracovaní.** Kompletný zoznam archívov je nasledujúci:

- Collection #1_Dumps - dehashed.tar.gz
- Collection #1_OLD CLOUD_Dump cleaned - deleted duplicated and trash.tar.gz
- Collection #1_EU combos_1.tar.gz
- Collection #1_OLD CLOUD_Gaming combos.tar.gz
- Collection #1_EU combos.tar.gz

[2] V preklade slovo Расшифровка značí rozšifrovanie

- Collection #1_OLD CLOUD_Hacking combos.tar.gz
- Collection #1_Games combos_Dumps.tar.gz
- Collection #1_OLD CLOUD_Japan combos.tar.gz
- Collection #1_Games combos_Sharpening.tar.gz
- Collection #1_OLD CLOUD_Monetary combos.tar.gz
- Collection #1_Games combos.tar.gz
- Collection #1_OLD CLOUD_OLD DUMPS DEHASHED.tar.gz
- Collection #1_MAIL ACCESS combos.tar.gz
- Collection #1_OLD CLOUD_Porn combos.tar.gz
- Collection #1_Monetary combos.tar.gz
- Collection #1_OLD CLOUD_Shopping combos.tar.gz
- Collection #1_NEW combo semi private_Dumps.tar.gz
- Collection #1_OLD CLOUD_Trading combos.tar.gz
- Collection #1_NEW combo semi private_EU combo.tar.gz
- Collection #1_OLD CLOUD_UK combos.tar.gz
- Collection #1_NEW combo semi private_Private combos.tar.gz
- Collection #1_OLD CLOUD_USA combos.tar.gz
- Collection #1_NEW combo semi private_Update Dumps.tar.gz
- Collection #1_RU combo.tar.gz
- Collection #1_Number pass combos.tar.gz
- Collection #1_Shopping combos.tar.gz
- Collection #1_OLD CLOUD_BTC combos.tar.gz
- Collection #1_USA combos.tar.gz

- Collection #1_OLD CLOUD_CHINA combos.tar.gz
- Collection #1_USER PASS combos.tar.gz

4.4 Spôsoby spracovania

Spracovanie údajov je možné riešiť troma rôznymi spôsobmi:

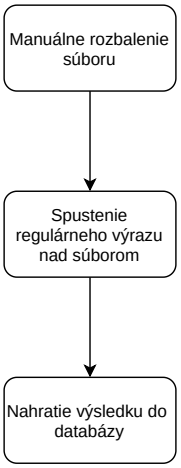
- Manuálne
- Automatizovane
- Poloautomatizovane

Na prvý pohľad sa to môže javiť nevýhodne, neriešiť jednotný prístup k spracovaniu údajov, ale treba brať do úvahy aj fakt, že každé riešenie má svoje pre a proti ako sa ukáže v nasledujúcich častiach.

Manuálne spracovanie

Manuálne spracovanie je aktuálny spôsob, akým sa uniknuté dáta spracovávajú vo firme REDAMP SECURITY s.r.o.. Proces spočíva v napísaní vhodného regulárneho výrazu, ktorý z rozbaleného súboru vyberie potrebné dáta a uloží ich do určeného súboru. Tento súbor je následne vstupný parameter python skriptu, ktorý sa pripojí k databázy a podľa určených argumentov vytvorí, alebo pridá hodnoty do vyžadovanej tabuľky.

Negatíva tohto prístupu sú zrejme. Hlavným je pomalá rýchlosť spracovania, nakoľko je celý proces manuálny a aj kvôli aktuálnym obmedzeniam návrhu v relačnej databáze nie je možné viacnásobné zapisovanie do tabuľky z dôvodu referenčnej integrity na úrovni RDBMS. Celý proces je znázornený na obrázku 4.13



Obr. 4.13: Proces manuálneho spracovania údajov [58]

Poloautomatizované spracovanie

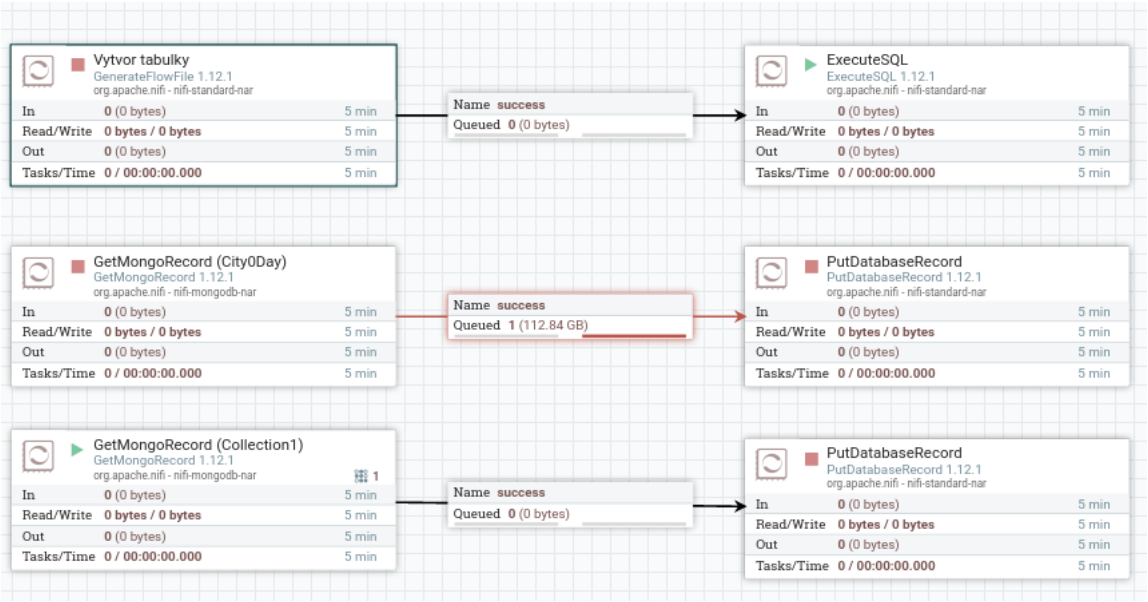
Poloautomatizované spracovanie spočíva vo využití externého nástroja (Apache NiFi) na rozbalenie, spracovanie a nahratie údajov do databázy. Medzi jeho výhody patrí možnosť využitia nástroja užívateľom, ktorý bol iba zaškolený v jeho používaní. Na rozdiel od manuálneho spracovania nie je potrebná hlboká znalosť systémových nástrojov a jazyka SQL a na rozdiel od automatického spracovania nie je pre zmenu potrebná znalosť programovania.

Z toho vyplýva, že je možné zaškoliť zamestnanca na prácu v prostredí programu (Apache NiFi), čo sa pozitívne odrazí na nákladoch, nakoľko nie je potrebné zamestnávať dátového analytika alebo programátora na spracovanie údajov. Po dostatočnom tréningu prácu s aplikáciou zvládne aj osoba bez technického vzdelania.

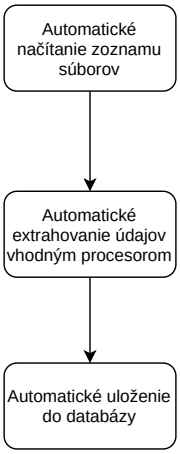
Nevýhodou však je mierne nižšia efektívna rýchlosť spracovania údajov, nakoľko existuje réžia spojená so spracovaním v programe, namiesto priameho spracovania manuálnym alebo automatickým spôsobom. Tento nedostatok sa však dá čiastočne kompenzovať spracovaním údajov v clusteri, kedy je možné škálovanie do šírky a taktiež zvolením správneho režimu v aplikácii Apache NiFi (flow based vs record oriented spracovanie)

Ďalšou nevýhodou je obmedzenie funkčnosti na procesory, ktoré sú oficiálne dostupné, čo môže v niektorých prípadoch limitovať spôsob spracovania údajov. Na druhú stranu je možné zmeniť databázový systém jednoducho zmenou procesora, do ktorého FlowFile smeruje. Príklad je na obrázku 4.14 a reprezentuje diagram, ktorý bol reálne využitý na konverziu MongoDB údajov do relačnej databázy pre záťažové testy.

Celý proces spracovania poloautmatizovaným spôsobom je znázornený na obrázku 4.15 a reálna ukážka využitia NiFi je popísaná v kapitole 5 a zobrazená na obrázku 5.2.



Obr. 4.14: Ukážka využitia NiFi na prenos údajov medzi NoSQL a relačnou databázou [58]



Obr. 4.15: Proces poloautomatizovaného spracovania údajov [58]

Automatizované spracovanie

Automatizované spracovanie využíva faktu, že veľká časť dostupných archívov má veľmi podobný formát. Je teda možné investovať čas a zdroje do skriptu, ktorý tento formát spracuje a poskytne dáta v podobe generátoru, ktoré sa dajú následne spracovať do vhodnej podoby pre uloženie vo zvolenej databáze.

Pre tento účel bola napísaná relatívne kompexná webová aplikácia, ktorá prostredníctvom Command Line Interface (CLI) očakáva cestu k archívom a následne každý archív automaticky rozbalí, nájde skript pomenovaný `normalization.py`, spustí ho a získa dáta v podobe generátoru.

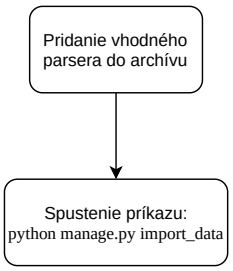
Tým je možné obísť problém s načítaním všetkých dát do pamäte, nakoľko sa údaje načítavajú iba po riadkoch a náročnosť na pamäť je minimálna. Čo sa týka náročnosti na miesto, tak je treba počítať s maximálnou veľkosťou archívu po rozbalení. Experimentovaním sa ukázalo, že je najvhodnejšie počítať s desaťnásobnou veľkosťou najväčšieho archívu pred jeho rozbalením.

Taktiež rýchlejší typ pamäte pre úložný priestor značne urýchli rozbaľovanie archívu a teda aj priepustnosť systému. Z toho dôvodu sa odporúča systém prevádzkovať na NVMe type pamäti, ktoré nie sú limitované rýchlosťou SATA rozhrania.

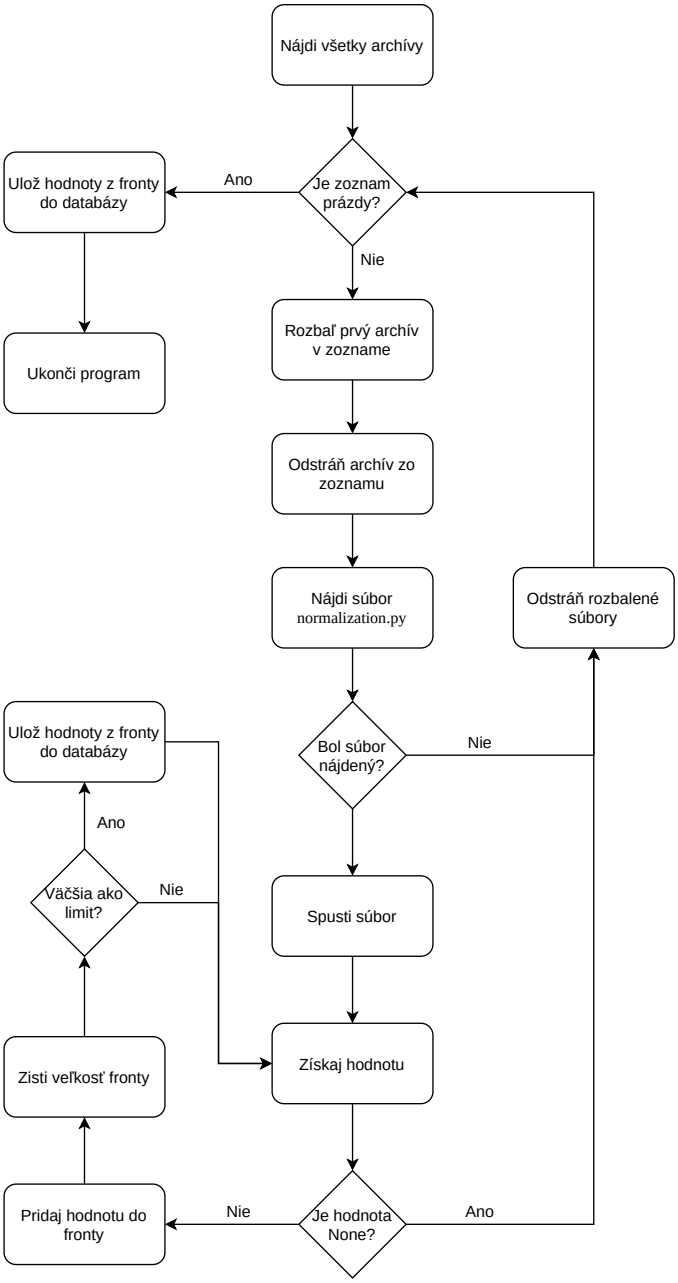
Ako ukázali testy, tento postup je zároveň najoptimálnejší. V prípade MongoDB systému je možné spracovať až 6 miliónov údajov za minútu a to v prípade sériového spracovania, pričom paralelné spracovanie je technický možné. Ukážku skriptu na spracovanie archívu je možné nájsť v prílohe D.

Výhodou je taktiež možnosť rýchleho a úplného obnovenia databázy zo zálohy, nakoľko dáta v archíve sa dajú považovať za spracované a štruktúrované. Nevýhodou je však potreba vlastného riešenia na ukladanie údajov do databázy, a napísanie skriptov, ktoré údaje spracujú a uložia. Z toho vyplýva, že automatizované spracovanie je vhodné využiť iba pri veľkom počte archívov s dopredu známou štruktúrou údajov.

Tento proces bol taktiež využitý na prípravu testovacej dátovej sady použitej v rámci testovania výkonu databázových systémov a detailné fungovanie je popísané na obrázku 4.17. Samotný proces spracovania údajov je znázornený na obrázku 4.16



Obr. 4.16: Proces automatického spracovania údajov



Obr. 4.17: Detailný proces fungovania automatizovaného importu dát [58]

4.5 Overenie výkonu data storage systémov

Výkon úložného (`data storage`) systému je kritickou súčasťou každého systému. Práve táto časť býva najčastejšie úzkym hrdlom (*bottleneck*) systému a jej náhrada za inú technológiu býva komplikovaná, alebo po technickej časti nerealizovateľná. Z toho dôvodu tvorí správna analýza databázového systému jadro každej architektúry, kde je dôležitý veľký objem spracovaných dát a transakcii.

Príprava testovacieho prostredia

Príprava testovacie prostredia spočívala v pripravení 3,2 miliard záznamov z wordlistu `Collection 1` a `City0Day`. Na tento účel boli využité dva spôsoby prípravy dát a to automatický a poloautomatický.

V prvej časti sa do každého jedného archívu vložil krátky skript, ktorý je možné vidieť v prílohe D a ktorý načítal všetky súbory v danom archíve a následne ich obsah premenil do podoby python slovníku, ktorý je štruktúrou podobný JSON-u. Tieto dáta sa následne automaticky načítali do jednej z predompripravených databáz a to buď PostgreSQL alebo MongoDB. Plne funkčné a optimalizované bolo hlavne vkladanie údajov do databázy MongoDB, pričom PostgreSQL bol použitý len v rámci hodnotenia výkonu systému Apache NiFi.

V ďalšej fázy nasledovalo poloautomatické spracovanie dát, kedy sa dáta prenášali medzi NoSQL (MongoDB) a relačnou (PostgreSQL) databázou prostredníctvom nástroja NiFi. Tento priebeh je graficky znázornený na obrázku 4.16.

Následne bolo potrebné nastaviť relačné a NoSQL databázové systémy pre optimálny výkon. Pre možnosť porovnania rýchlosti boli vytvorené identické linux kontajneri (LXC) s nasledujúcimi parametrami:

- 8 x virtual CPU (vCPU)
- 8GB RAM
- SSD disk podľa potreby

Tieto LXC kontajneri bežali v rámci nevyužívaného uzlu v Proxmox clusteri, ktorý sa v rámci spoločnosti REDAMP SECURITY s.r.o. využíva iba na testovacie účely. Týmto spôsobom neboli vo väčšine prípadov výsledky zaťažené zdieľaním zdrojov medzi viacerými, na výkon a prostriedky náročnými službami. Po hardwardovej stránke má uzol clusteru nasledujúce parametre:

- 4x SSD CRUCIAL MX 500[3] v RAID10
  - SATA III
  - Náhodne čítanie: 95 000 IOPS
  - Náhodný zápis: 90 000 IOPS
  - Rýchlosť čítania: 560 MB/s
  - Rýchlosť zápisu: 510 MB/s
- 24 x Intel(R) Xeon(R) CPU E5-2430L v2 @ 2.40GHz (2 Sockets)
- HPE SmartMemory 192GB DDR3 16000 MHz

Optimalizovanie vkladania údajov pre relačné databázy

Optimalizácia vkladania dát je komplikovaná oblasť, ktorá je špecifická od údajov a spôsobu, akým sa k údajom pristupuje. Ako sa ukázalo počas implementácie, je dôležité zvoliť správny spôsob vkladania dát, nakoľko to môže byť rozdiel medzi rýchlosťou tisíc položiek za sekundu a milión.

Možno ani nie tak prekvapivo (nakoľko NoSQL databázy majú na prvotné vkladanie veľkého objemu dát vlastné API) sa tento problém týkal prevažne relačných databáz.

Ukázalo sa, že vkladanie pomocou bežného príkazu INSERT je veľmi pomalé, pričom rýchlosť bola v jednotkách tisícoch a rýchlosť degradovala s počtom záznamov v databáze. To bolo spôsobené referenčnou integritou v podobe unikátnych indexov, ktoré sa museli prechádzať počas vkladania údajov, nakoľko základný typ indexu v prípade databázy PostgreSQL je B-tree [63], časová náročnosť prehľadávania indexu bola v najlepšom prípade logaritmická.

Tento problém sa dal vyriešiť odstránením unikátnych indexov, avšak stále bolo prehľadávanie pomalé z dôvodu prepočítavania indexu pri každom novom zázname. Ku koncu sa pristúpilo k úplnému odstráneniu indexu a jeho vytvoreniu až na konci operácie. Rýchlosť sa týmto spôsobom vrátila znovu na hodnoty v tisícoch vložení za sekundu, avšak v prípade malého vzorku o veľkosti troch miliard záznamov by vloženie všetkých potrebných údajov trvalo v prípade vloženia troch tisíc záznamov za sekundu[4] približne jedenásť dní, čo je nedostatočné. Zároveň v prípade vytvorenia indexu na konci operácie nie je možné vytvoriť unikátny index.

[3] `https://www.crucial.com/ssd/mx500/CT500MX500SSD1`

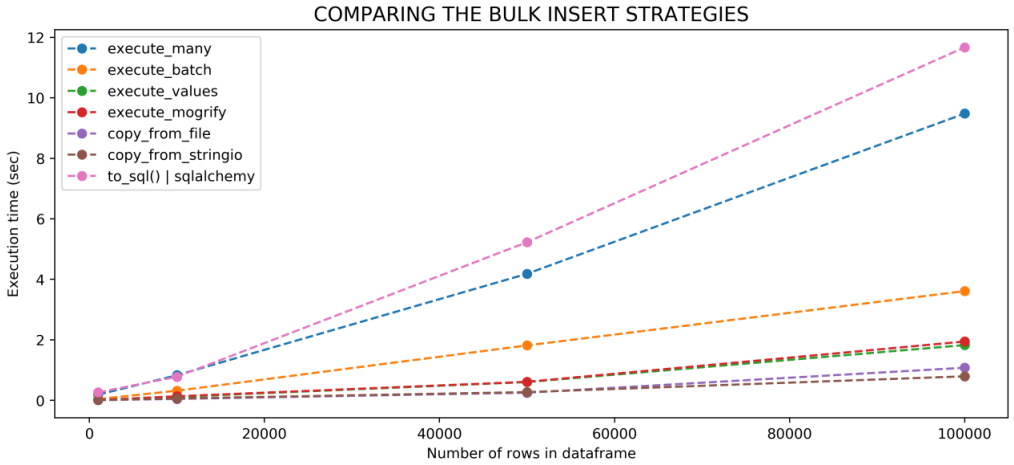
[4] Údaje nie sú uvádzané presne, nakoľko rýchlosť vkladania je závislá od viacerých faktorov, akými je IO load zariadenia, CPU a typ úložnej pamäte. Presné hodnoty nie sú dôležité a vystačia hodnoty v rádoch (tisíce, milióny a pod.)

Funkcia	Čas (s)	Pamäť (MB)
insert_one_by_one()	128.8	0.08203125
insert_executemany()	124.7	2.765625
insert_executemany_iterator()	129.3	0.0
insert_execute_batch()	3.917	2.50390625
insert_execute_batch_iterator(page_size=1)	130.2	0.0
insert_execute_batch_iterator(page_size=100)	4.333	0.0
insert_execute_batch_iterator(page_size=1000)	2.537	0.2265625
insert_execute_batch_iterator(page_size=10000)	2.585	25.4453125
insert_execute_values()	3.666	4.50390625
insert_execute_values_iterator(page_size=1)	127.4	0.0
insert_execute_values_iterator(page_size=100)	3.677	0.0
insert_execute_values_iterator(page_size=1000)	1.468	0.0
insert_execute_values_iterator(page_size=10000)	1.503	2.25
copy_stringio()	0.6274	99.109375
copy_string_iterator(size=1024)	0.4536	0.0
copy_string_iterator(size=8192)	0.4596	0.0
copy_string_iterator(size=16384)	0.4649	0.0
copy_string_iterator(size=65536)	0.6171	0.0

Tabuľka 4.1: Tabuľka rýchlosti vkladania údajov [46]

V tabuľke 4.1, je možné vidieť niekoľko rôznych a optimálnejších spôsobov, ktoré je možné použiť pre vkladanie väčšieho množstva údajov. Pre naše účely bola využitá metóda `stringio()`, ktorá využíva funkciu COPY a ktorá očakáva vstup zo štandardného vstupu (*stdin*). Pamäťová náročnosť je irelevantná, nakoľko vďaka vstupu údajov z generátora je možné množstvo dát na vstupe limitovať podľa potrieb.

Ako je možné vidieť na obrázku 4.18, táto metóda je najoptimálnejšia zo všetkých možností a to hlavne zo stúpajúcim množstvom dát. V samotnej testovacej implementácii je taktiež implementovaný zásobník (*buffer*) pre obmedzenie množstva údajov v pamäti.



Obr. 4.18: Porovnanie výkonu metód vkladania údajov v PostgreSQL [60]

Optimalizovanie konfigurácie

Optimalizácia data storage technológii patrí k ďalším dôležitým parametrom. V základnej konfigurácii napríklad PostgreSQL počíta s behom na rotačných diskoch [48] a tomu má upravené váhy pre výpočet najoptimálnejšieho prístupu k údajom. Nasledujúce parametre preto boli zmenené v základnej konfigurácii PostgreSQL 13:

- `effective_io_concurrency = 400`
- `wal_compression = on`
- `max_wal_size = 1GB`
- `min_wal_size = 80MB`
- `random_page_cost = 1.1`

Hodnota `effective_io_concurrency` udáva počet paralelných operácii, ktoré pristupujú k disku v rámci PostgreSQL. V prípade magnetických diskov je odporúčaná hodnota rovná počtu diskov v RAID 0 alebo RAID 1 poli [13]. V prípade aktuálnej konfigurácie, kde pracujeme s SSD diskami, je odporúčaná hodnota 200 [45], pričom z dôvodu, že SSD disky bežia v RAID10, môžeme zvoliť až hodnotu 400.

Táto konfigurácia zrýchľuje prevažne čítanie hodnôt, nakoľko paralelné zapisovanie do databázy nie je vo väčšine prípadov možné z dôvodov ACID vlastnosti. Rozdiel v rýchlosti čítania pri rôznych nastaveniach je možné vidieť v tabuľke 4.2.

effective_io_concurrency	Average runtime in ms
1	65604
20	17807
100	16776
1000	16713

Tabuľka 4.2: Priemerná doba behu pre rôzne hodnoty `effective_io_concurrency` [33]

Základná hodnota `shared_buffers` je pre aktuálne parametre LXC kontajnera dostačujúca. Treba však zvýšiť hodnoty `max_wal_size`, `min_wal_size` a `wal_compression`, čo zabezpečí väčšie zaťaženie procesora a zníži množstvo zápisov na disk, nakoľko dáta z WAL sa zapisujú na disk menej často. Tieto parametre zvýšia rýchlosť zápisu na disk [45].

Parameter `random_page_cost` patrí pravdepodobne medzi tie najdôležitejšie. Jeho hodnota by mala byť v rozmedzí 1.1 až 1.5 pre SSD disky a Non-Volatile Memory (NVMe) pamäte a hodnota 2 pre RAID pole. Táto hodnota priamo ovplyvňuje plánovač dotazov a zvyšuje tak cenu náhodného čítania a zápisu z databázy v porovnaní so sekvenčným čítaním a zápisom [62]. Hodnota by však nemala byť rovná jedna, nakoľko by to znamenalo, že cena čítania a zápisu z disku je rovná cene čítania a zápisu z RAM.

V prípade použitia výkonnejšieho LXC kontajnera, alebo pri použití dedikovaného hardware je vhodné zmeniť základné parametre nasledujúcich parametrov:

- `maintenance_work_mem`
- `work_mem`
- `checkpoint_segments`
- `max_worker_processes`
- `max_parallel_workers`

Čo sa týka konfigurácie MongoDB, tá je dostačujúca. Jediný parameter, ktorý bolo potrebné zmeniť, je zapnutie kompresie, ktoré nie je pri čistej inštalácii zapnuté. Na výber je niekoľko rôznych druhov kompresii, z ktorých najlepšie vychádza kompresný algoritmus `bzip2` v podobe knižnice `zlib`. Jej zapnutie sa robí v podobe upravenia konfiguračného súboru pridaním nasledujúcich riadkov:

```
storage:
  wiredTiger:
    collectionConfig:
      blockCompressor: "zlib"
```

Listing 2: Nastavenie MongoDB pre používanie kompresie [58]

Základná konfigurácia Greenplum je dostatočná. V prípade identifikovaných problémov s nastaveniami samotná aplikácia upozorní na chybné hodnoty a parametre. Je však potrebné upozorniť, že Greenplum svoje prednosti ukáže hlavne pri behu v clusteri. Nakoľko operácie, ktoré sa v rámci testovania robia, sú náročné na fyzické zdroje, uzly Greenplum clusteru neboli všetky na rovnakom Proxmox uzly. Každý Greenplum uzol sa nachádzal na fyzicky rozdielnom servery.

Vkladanie údajov do MongoDB

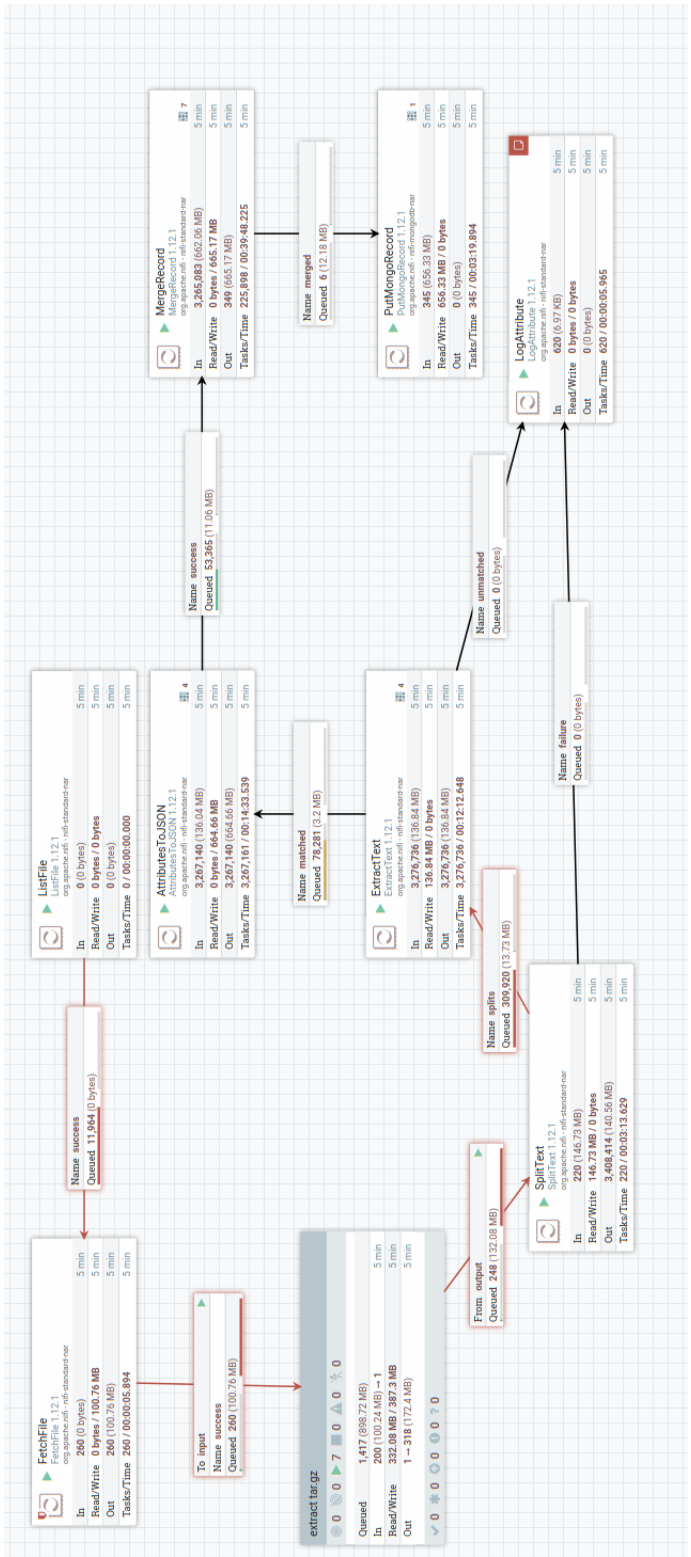
Pre vkladanie údajov do databázového systému MongoDB bol naprogramovaný vlastný skript, ktorý rozbalil každý jeden archív, vyhľadal v ňom súbor `normalization.py` a spustil ho. Tento vlastný skript identifikoval všetky textové súbory v danom archíve, prečítal ich a výsledky postupne vrátil hlavnému skriptu. Tým sa zaručila minimálna pamäťová náročnosť a taktiež možnosť paralelného spracovania viacerých záznamov naraz.

Okrem vlastného riešenia bolo na tento účel otestované aj použitie nástroja Apache NiFi, ktoré však ako bolo pomalé. Pozitívnym ukazovateľom je však fakt, že spomalenie bolo na úrovni CPU, ktoré by odstránila dodatočná paralelizácia a nie na úrovni disku, ktorého odstránenie by bolo problematické. Spracovanie pomocou nástroja Apache NiFi je možné vidieť na obrázku 4.19. Na rozdiel od záťažového testovania, tento export bol robený na zariadení s nasledujúcimi parametrami:

- Intel® Core™ i7-6700K CPU @ 4.00GHz × 8
- HyperX 32GB DDR4 2400MHz CL15
- 1x Samsung 970 EVO 1TB
  - SSD disk M.2 (PCIe 3.0 4x NVMe)
  - Náhodné čítanie: 500 000 IOPS
  - Náhodný zápis: 450 000 IOPS
  - Rýchlosť čítania: 3 400 MB/s
  - Rýchlosť zápisu: 2 500 MB/s

Ako je z parametrov možné vidieť, na rozdiel od SSD disku použitého pri LXC kontajneroch, ktorý je pripojený prostredníctvom rozhrania SATA3 s teoretickou maximálnou rýchlosťou 750 MB/s (reálna 560 MB/s čítanie a 510 MB/s zápis), tak použitý NVMe disk zvláda teoreticky štvornásobnú rýchlosť čítania a zápisu a päťnásobnú rýchlosť pri počte náhodných operácii za sekundu. Parametre výkonu a frekvencia procesora sú v tomto prípad tiež výkonnejšie oproti použitému serverovému CPU. Navyše sa jedná o dedikované zariadenie, na ktorom nebežali v čase exportu žiadne vedľajšie úlohy.

Dáta na konečné zariadenie boli prenesené spôsobom vytvorenia zálohy a jej obnovenia na výslednom zariadení.



Obr. 4.19: Vkladanie údajov do MongoDB prostredníctvom aplikácie Apache NiFi [58]

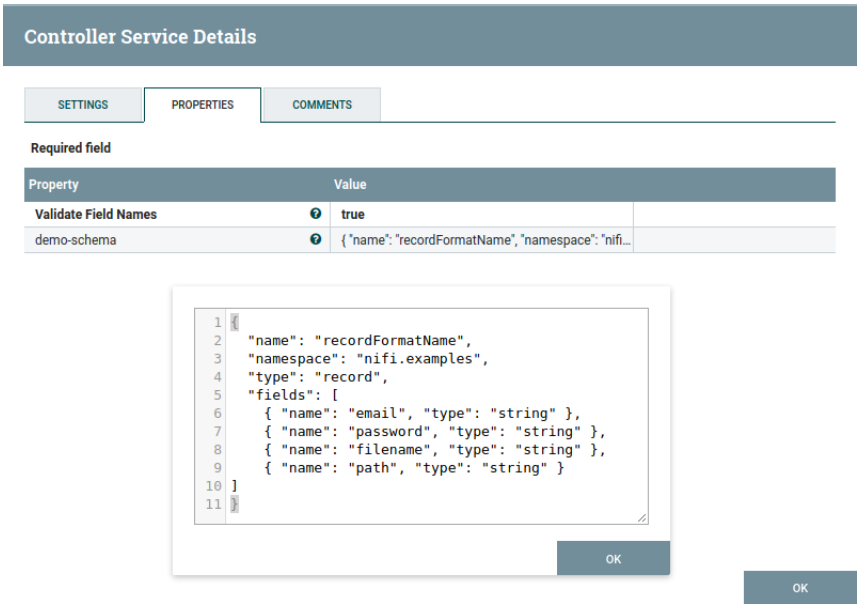
Vkladanie údajov do PostgreSQL

Na vkladanie údajov do databázového systému PostgreSQL bol využitý nástroj Apache NiFi a vytvorené tri hlavné procesy. V prvom procese sa vytvoril prázdny FlowFile, ktorý slúžil ako spúšťač procesu `ExecuteSQL`. Ten vytvoril potrebné tabuľky s parametrami. V ďalších dvoch procesoch `GetMongoRecord` sa údaje načítali z MongoDB a následne sa cez procesor `PutDatabaseRecord` nahrali do databázy. Tento postup bol vytvorený špecificky za účelom naplnenia relačnej databázy PostgreSQL z NoSQL databázy MongoDB.

Údaje sa prenášali v rámci Apache NiFi vo formáte Avro, ktorý je popísaný v sekcii 3.3. Pre tento účel bolo potrebné vytvoriť tri kontrolérové služby (controller services) a to `AvroReader`, `AvroRecordSetWriter` a `AvroSchemaRegistry`.

Kontrolér `AvroRecordSetWriter` sa využíva na uloženie dát z NoSQL databázy a to v v prípade procesoru `GetMongoRecord`, kde sa špecifikuje `Record writer` na `AvroRecordSetWriter`. V tomto kontroléri sa následne určí `Schema registry`, pre ktorú sa vytvorí `AvroSchemaRegistry` zobrazená na obrázku 4.20. Tu sa definujú parametre získaných údajov, rovnako ako aj návratové dátové typy. Následne sa v procesore `PutDatabaseRecord` špecifikuje `Record Reader` na `AvroReader`, typ operácie na `INSERT` a určí sa služba na pripojenie k databáze. V prípade PostgreSQL táto služba nie je v základnej inštalácii a je potrebné ju nainštalovať z oficiálnych stránok projektu PostgreSQL a presunúť do vhodnej zložky v rámci Apache NiFi.

Výsledný tok dát je možné vidieť na obrázku 4.14. Všetky procesory sú `RecordOriented`, čím sa síce odstraňuje časť abstrakcie (tým, že sa špecifikuje, o aké dáta sa jedná). Na druhú stranu je možné zvýšiť priepustnosť systému pri veľkom počte malých údajov, nakoľko sa tým obchádza ukladanie na disk v neefektívnej forme.



Obr. 4.20: Ukážka definovania schémy pre formát Avro [58]

Vkladanie údajov do Greenplum

Ako bolo spomenuté v sekcii 3.4, tak Greenplum vychádza z PostgreSQL verzie 8 a vďaka tomu je kompatibilný so syntaxou a databázovou zálohou PostgreSQL. Nakoľko sa jedná o pomerne neznámu technológiu, tak v čase písania diplomovej práce nebol k dispozícii procesor alebo kontrolér, ktorý by umožňoval pripojenie a vkladanie dát cez Apache NiFi do Greenplum.

Tento problém sa vyriešil nainštalovaním PostgreSQL verzie 8.2. Následne sa spravil export dát z MongoDB do PostgreSQL 8.2 cez Apache NiFi a to identicky, ako je to popísané v sekcii 4.5.

Z tejto verzie PostgreSQL sa následne spravila databázova záloha príkazom `pg_dump`. Túto zálohu by bolo možné importovať na priamo do Greenplum, avšak nevyužívala by plný potenciál stĺpcovo orientovanej databázy a údaje by neboli rozdistribuované medzi jednotlivé uzly. Pre tento účel bolo potrebné zálohu rozbaliť. Tu však nastával problém, nakoľko sa ukázalo, že žiaden textový alebo grafický editor si neporadí s editovaním súboru o veľkosti stoviek gigabajtov.

Riešením teda bolo využitie štýlu, akým databázová záloha vznika a teda faktu, že všetky definície tabuliek a ich vlastnosti sú na začiatku výstupného súboru. Príkazom `head -n XXX > head.sql` sa odstrihla časť, v ktorej sa vytvárala databáza a tabuľky, ktoré sa upravili z pôvodnej podoby na podobu zobrazenú vo výpise 3. V časti `WITH` sa špecifikuje, že sa jedná o stĺpcovo orientovanú databázu, že sa má použiť kompresia (ktorá je v PostgreSQL obmedzená iba na záznamy a nie na

celé stĺpce) a že sa údaje majú rozdistribuovať náhodne. Tým, že nie je určený primárny kľúč, sa automaticky údaje rozdistribuujú podľa `id`.

```
CREATE TABLE city0day (
    id bigint NOT NULL,
    email text,
    "password" text,
    filename text,
    path text
) WITH
  ↪ (appendonly=true,compresstype=zlib,orientation=column,compresslevel=5)
  ↪ DISTRIBUTED RANDOMLY;
```

Listing 3: SQL príkaz `CREATE` pre Greenplum [58]

4.5.1 Výsledky záťažových testov

Tak ako pri všetkých štatistikách a porovnaniach, aj nasledujúce výsledky záťažových testov sa dajú čítať rôzne. Pri tvorbe štatistík sa vychádzalo z dvoch základných otázok, na ktoré bolo treba získať odpoveď:

- Porovnanie výkonu Greenplum a PostgreSQL
- Porovnanie MongoDB a PostgreSQL

Dve základné vlastnosti, ktoré nás v tomto prípade zaujímajú sú:

- Rýchlosť prístupu k informáciám
- Výsledná veľkosť uložených dát

Pravda porovnávanie rôznorodých tried databázových technológii (relačné, stĺpcovo orientované, key-value) treba brať kriticky. Ako príklad je možné uviesť, že sa porovnáva PostgreSQL a MongoDB, ktoré bežia v jednej inštancii s technológiou Greenplum, ktorá beží na troch uzloch a troch fyzických zariadeniach[5].

Zároveň každá technológia má svoje silné a slabé stránky, ktoré často vyplývajú zo samotnej základnej architektúry pre danú triedu. Nedá sa teda jednoznačne povedať, ktorá je najlepšia (parametrov, podľa ktorých by sme to mohli porovnávať je veľa). Z toho dôvodu budú silné a slabé stránky každej technológie zhrnuté na záver a to z pohľadu implementácie vo finálnej architektúre.

Testy, ktoré boli napísané a použité na porovnanie výkonu, sú uvedené v prílohe A, B a C.

[5] Kde je to potrebné, boli údaje normalizované

Rýchlosť prístupu k informáciám pre základné operácie

Porovnávanie výkonu prístupu k informáciám sa robilo na základných SQL operáciach:

- `COUNT`
- `INNER JOIN` (`SELECT`)
- `LIKE`[6]
- `DISTINCT`

Značný rozdiel môže spraviť zapnuté indexovanie nad správnym stĺpcom. Z toho dôvodu bol vytvorený základný index nad stĺpcom `email`. V prípade technológie Greenplum a PostgreSQL sa jedná o implementáciu B-tree algoritmu. V prípade MongoDB sa jedná o typ algoritmu hash map. Tieto hodnoty, získavané pri vytvorenom idexe, sú v grafe označené hviezdičkou.

Nakoľko MongoDB nemá podporu jazyka SQL, prešlo sa na alternatívne volanie príkazov prostredníctvom linuxovej aplikácie `mongo`, ktorá sa volala lokálne na danej inštancii. Problematické to môže byť hlavne v prípade `INNER JOIN`, ktorý nemá alternatívu v MongoDB. Namiesto toho sa zavolala dvakrát alternatíva k príkazu `SELECT`, ktorý však nemôže profitovať z paralelného prístupu k čítaniu a optimalizácii na úrovni jadra danej technológie (query planner v prípade RDBMS)

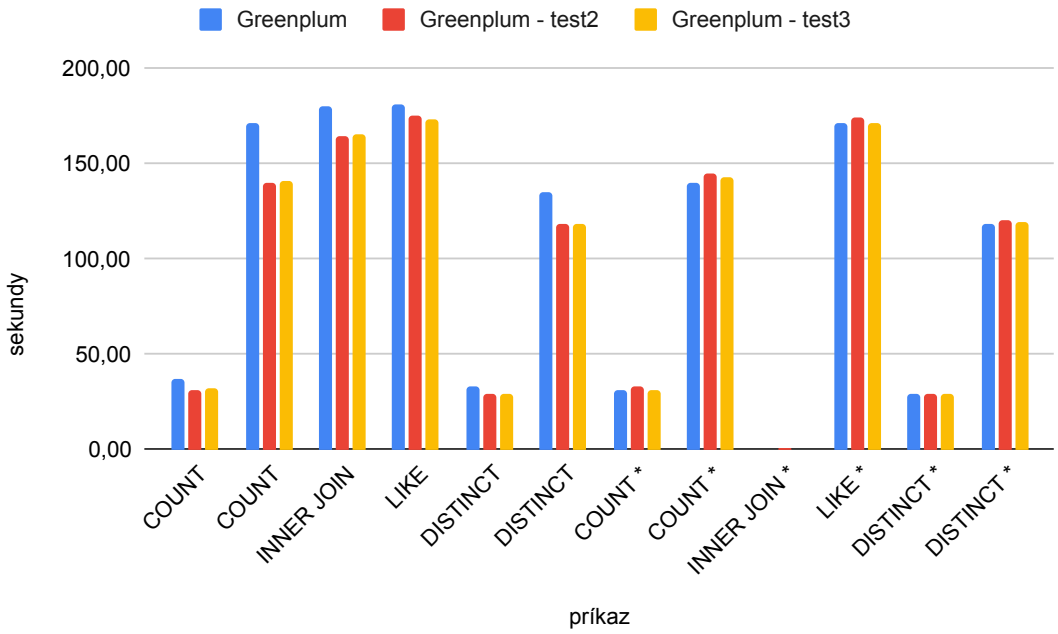
Z dôvodu, že Greenplum beží v clusteri troch inštancií, pričom iba jeden je prevádzkovaný na rovnakom fyzickom serveri ako ostatné technológie (po hardware stránke sú serveri identické), bolo potrebné počítať aj s výkyvmi na úrovni prístupu ku zdrojom. Pre tento účel sa záťažový test opakoval trikrát a informácie boli použité z prvého behu záťažového testu.

Ako je možné vidieť na obrázku 4.21, tak aj v priebehu neskorších opakovaní, ktoré boli robené v rozmedzí jedného týždňa a medzi dvoma testami, uplynulo najmenej 24 hodín, nebol výkyv veľmi drastický.

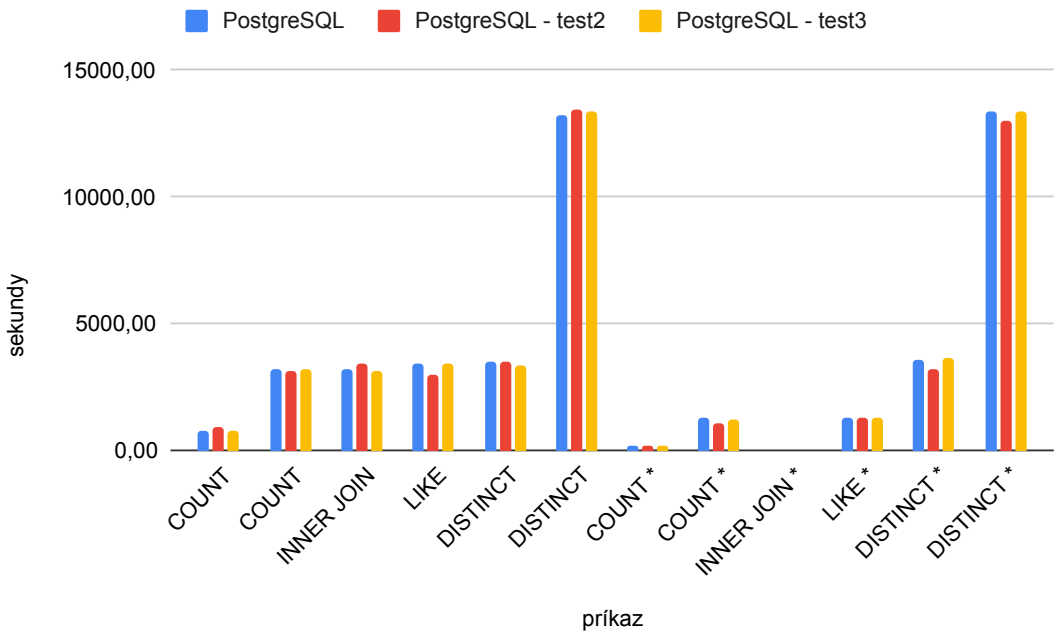
Rovnaký test bol použitý aj na overenie stability prístupu k zdrojom pre technológie, ktoré bežali iba na jedinom LXC kontajneri. Výsledky pre technológiu PostgreSQL sú zobrazené na obrázku 4.22 s tým, že výsledky pre MongoDB boli podobné a z dôvodu prehľadnosti nie sú uvedené. Je vhodné taktiež uviesť fakt, že testy pre jednotlivé databázové technológie nebežali paralelne, nakoľko by výsledky mohli byť skreslené prístupom ku zdrojom (hlavne IO).

Pre porovnanie výkonu PostgreSQL a Greenplum boli údaje pre technológiu PostgreSQL trojnásobné zmenšené. Vychádza sa z predpokladu, že Greenplum beží trojnásobne rýchlejšie (vďaka

[6] V tomto prípade na najdrahšej operácii `%hladany_vzor%`



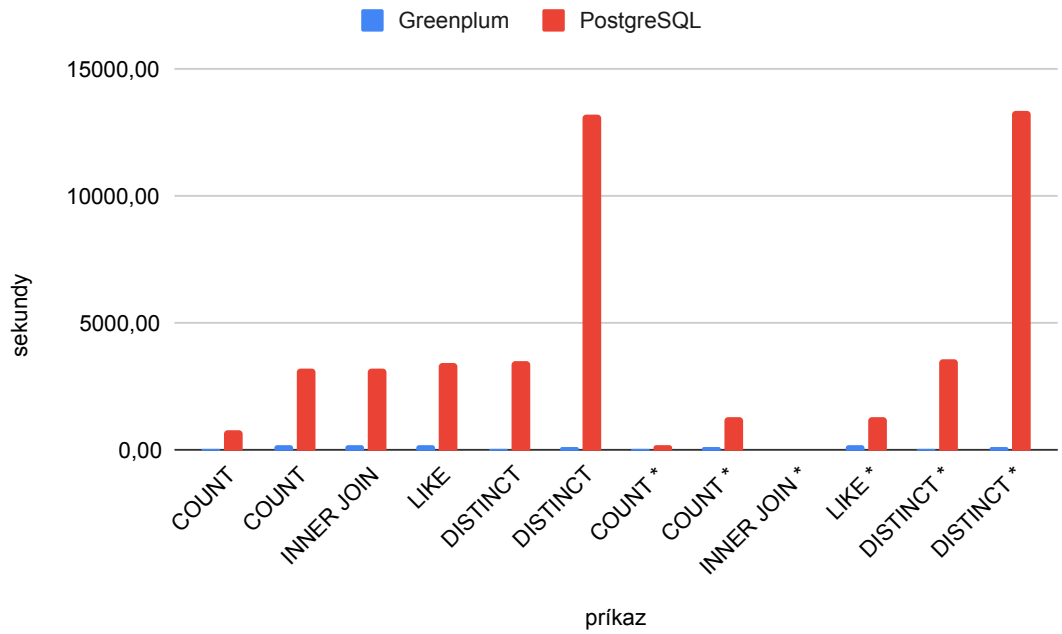
Obr. 4.21: Porovnanie výsledkov pri opakovaní testu nad technológiou Greenplum [58]



Obr. 4.22: Porovnanie výsledkov pri opakovaní testu nad technológiou PostgreSQL [58]

trom inštanciám) a týmto spôsobom sú údaje zo záťažového testu normalizované tak, aby ich bolo možné porovnávať. Pravda jedná sa o veľmi primitívne riešenie, ktoré neberie do úvahy napríklad fakt, že aj v prípade, že by mala technológia PostgreSQL trojnásobne výkonnejší hardware

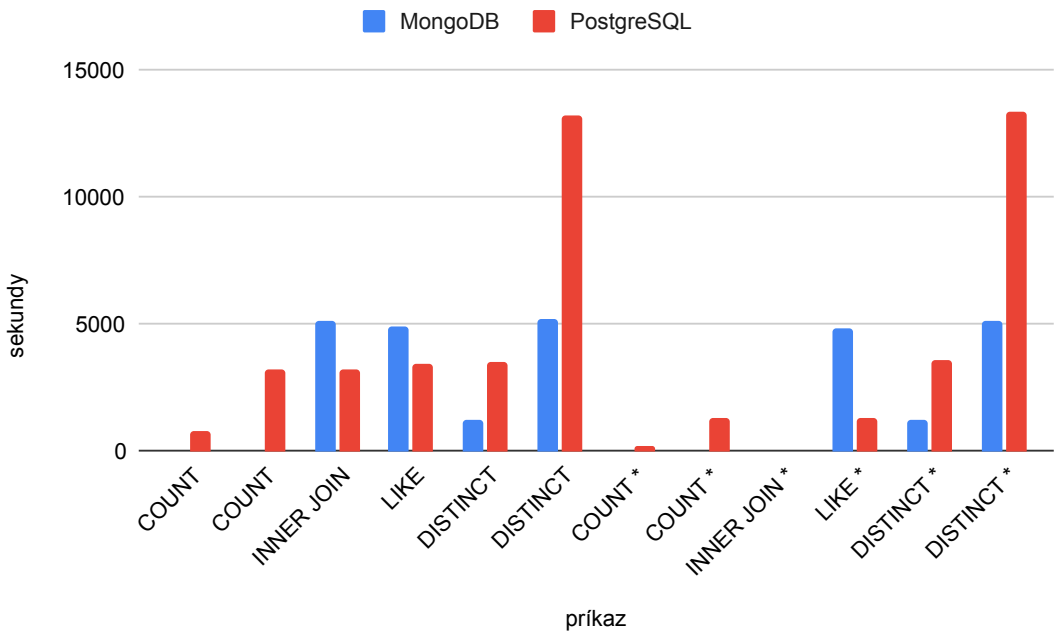
(vertikálne škálovanie), stále by bola limitovaná množstvom IO operácii, ku ktorým dokáže SSD disk pristupovať. Ako je možné vidieť na obrázku 4.23, tak rozdiel vo výkone je stále drastický. V prípade potreby je možné nájsť nenormalizované hodnoty na obrázku 4.25.



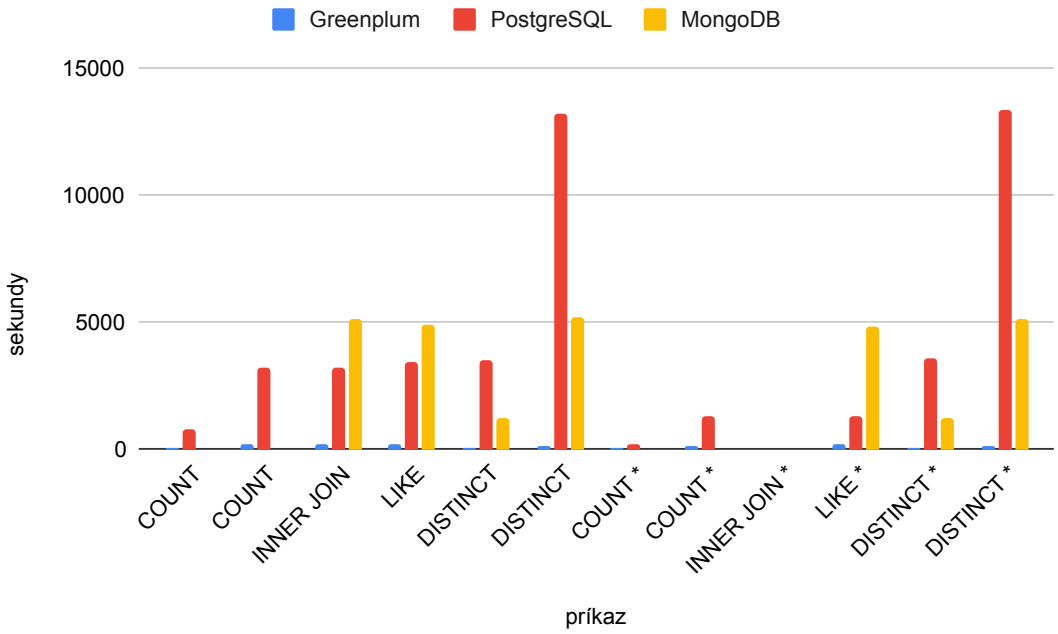
Obr. 4.23: Porovnanie výkonu technológie Greenplum a PostgreSQL (normalizované) [58]

Ako je možné vidieť na obrázku 4.24, pri porovnaní technológie PostgreSQL a MongoDB niektoré operácie trvajú dlhšie v MongoDB ako PostgreSQL. Špecificky operácia `LIKE`[7] je v prípade MongoDB pomalšia. Možným vysvetlením je pravdepodobne to, že regulárne výrazy majú väčšiu funkcionalitu ako `LIKE`, a preto sú komplikovanejšie na implementáciu[8]. Na druhú stranu nájdenie unikátnych údajov je značne rýchlejšie v MongoDB. Grafy jednotlivých technológii je možné nájsť v prílohe E.

[7] MongoDB alternatíva má podobu `.*hladany_vzor.*`
[8] V SQL existuje alternatíva `REGEXP`



Obr. 4.24: Porovnanie výkonu technológie Greenplum a PostgreSQL [58]



Obr. 4.25: Spoločné porovnanie výkonu všetkých technológii (údaje pre PostgreSQL nie sú normalizované) [58]

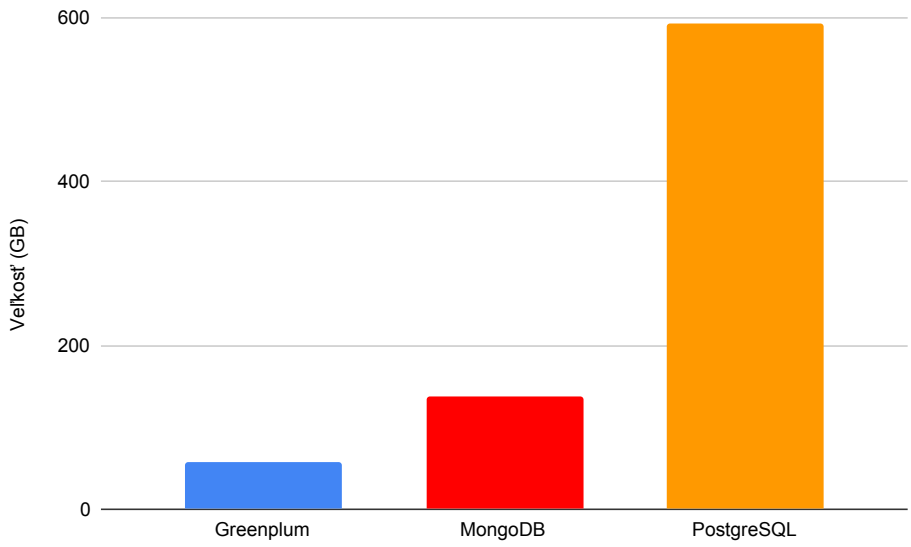
Výsledná veľkosť uložených dát

Výsledná veľkosť uložených dát nemusí zodpovedať veľkosti samotných dát. V tomto prípade nás zaujíma, akú veľkosť dáta reálne zaberajú na disku. Tento parameter ovplyvňujú:

- Kompresia
- Indexy

V prípade výsledkov na obrázku 4.26 sa do úvahy nebrali žiadne indexy, ktoré by bolo potrebné vytvoriť manuálne. To vysvetľuje rozdiel v úložnom priestore pre technológie Greenplum a MongoDB, ktoré používajú rovnaký kompresný algoritmus (`bzip2`), avšak na rozdiel od Greenplum má MongoDB povinný jeden index s názvom `_id`, ktorý identifikuje zvolený záznam v databáze (viď vlastnosti key-value databázy).

Najhoršie z porovnania vychádza technológia PostgreSQL, ktoré má kompresiu iba na úrovni riadkov a tá je v aktuálnom prípade neúčinná.



Obr. 4.26: Porovnanie veľkosti uložených informácii pre PostgreSQL, MongoDB a Greenplum [58]

4.5.2 Klady a zápory jednotlivých technológii

V tejto sekcii sú zhrnuté klady a zápory jednotlivých technológii. Neexistuje technológia, ktorá by bola dokonalá v každom ohľade, a preto je vhodné vyzdvihnúť klady a zápory a to hlavne z dôvodu aplikácie týchto znalostí v ďalších projektoch.

Greenplum

Medzi klady tejto technológie patrí hlavne kompatibilita so staršími verziami PostgreSQL. Vďaka tomu je možné zabezpečiť analytikom prístup k údajom v podobe jazyka SQL. Medzi pozitívne stránky taktiež patrí možnosť kompresie údajov jedným z najoptimálnejších algoritmov (rýchlosť vs úroveň kompresie) v dnešnej dobe a to `bzip2`.

Greenplum bol špecificky navrhnutý na uloženie a spracovanie veľkého objemu údajov a k tomuto účelu má upravenú architektúru. Ako bolo dokázané v aktuálnej kapitole, pri prístupe k informáciám pre OLAP výkonovo jasne vedie Greenplum nad technológiou PostgreSQL. Tomu odpovedá aj jednoduché horizontálne škálovanie, ktoré je v PostgreSQL problematické. Medzi ďalšie výhody patrí splnenie ACID vlastností a silná paralelizácia.

Medzi nevýhody by sa dalo zaradiť problematické nasadenie (v prípade tejto práce to zabralo tri dni) a nemožnosť úpravy uložených záznamov. Na tento účel je potrebné daný záznam zmazať a znovu nahrať. Zároveň cudzie kľúče nemôžu byť použité na vynútenie referenčnej integrity údajov. Je ich však možné využiť na vytvorenie väzieb medzi údajmi. Zároveň nie je možné využiť hash map ako typ indexu, ktorý má na rozdiel od B-tree konštantnú časovú náročnosť. Základný index B-tree má časovú náročnosť lineárnu, čo môže byť problematické pri väčšom objeme údajov (predlžuje sa doba prehľadávania). Na druhú stranu hash map je možné použiť iba na porovnanie a nie na aritmetické operácie (väčšie, menšie, rovná sa), ako je tomu pri B-tree.

Silné stránky:

- Kompresia uložených údajov prostredníctvom `bzip2`
- Schopnosť uložiť dáta distribuovane
- ACID
- Paralelné spracovanie údajov
- OLAP zameranie namiesto OLTP

Slabé stránky:

- Komplikovanejšie nasadenie
- Bez možnosti editovania existujúcich záznamov
- Referenčná integrita prostredníctvom FOREIGN KEY nemôže byť vynútená
- Bez podpory indexu HASH (hash map algoritmus) pre stĺpcovo orientovanú tabuľku
- Nemožnosť vytvárať indexy paralelne

PostgreSQL

PosgtreSQL je overená technológia, ktorú je možné v prípade dôslednej konfigurácie nastaviť tak, aby pracovala s údajmi o veľkosti niekoľkých terabajtov. Má veľkú komunitu a podporu aj pre špecifické projekty, akým je napríklad geographic information system (GIS). Na druhú stranu aktuálne neexistuje podpora stĺpcovo orientovaného úložiska a teda aj kompresie na úrovni stĺpcov, čo údaje v databáze značnej nafukuje. Z toho dôvodu je technológia PostgreSQL vhodná hlavne na prácu v OLTP systémoch. Na implementácii stĺpcovo orientovanej technológie sa aktuálne pracuje v projekte Zed store (viď 3.4), čo umožní opätovné zosynchronizovanie projektov Greenplum a PostgreSQL.

Silné stránky:

- Veľká komunita
- Vertikálne škálovanie
- Referenčná integrita
- Možnosť vytvárania indexov paralelné

Slabé stránky:

- Zameranie na OLTP
- Bez možnosti kompresie na úrovni stĺpcov
- Bez možnosti horizontálneho škálovania

MongoDB

Technológia MongoDB vyšla z testu najlepšie. Je porovnateľne rýchla s PostgreSQL, horizontálne škálovateľná a pritom je možné veľkosť redukovať zapnutím kompresie. Natívna podpora REST API umožňuje rýchlu tvorbu webového rozhrania nad údajmi a podpora indexovacie algoritmu hash map robí identifikovanie vhodných údajov rýchlym (konštantná časová náročnosť). Zároveň, nakoľko sa jedná o NoSQL bezschémovú databázu, nie je potrebné analyzovať dáta pred ich samotným uložením a obsah samotných záznamov môže byť rôzny. Medzi jej nevýhody patrí chýbajúca referenčná integrita a relácie medzi údajmi (t.j. nemá ACID vlastnosti). Zároveň nie je možné pristupovať k údajom prostredníctvom jazyka SQL (možný popis prístupu cez SQL je popísaný v

kapitole 5.4).

Silné stránky:

- Kompresia prostredníctvom `bzip2`
- Distribuované uloženie dát
- Paralelné spracovanie údajov
- Horizontálne škálovanie
- Podpora indexovacieho algoritmu hash map
- So zameraním na REST API

Slabé stránky:

- Bez ACID vlastností
  - Bez referenčnej integrity
  - Bez relácii medzi údajmi
- Bez priamej podpory SQL

Kapitola 5

Vlastný návrh a implementácia

Návrh architektúry pre efektívnu analýzu bezpečnostných dát sa skladá z niekoľkých častí, ktoré sú z dôvodu prehľadnosti popísané samostatne, a následne zlúčené v poslednej sekcii danej kapitoly. Návrh si dáva za cieľ efektívnu analýzu údajov, s možnosťou prístupu pre analytikov a s možnosťou rôznych prístupov k spracovaniu údajov.

Dôležitým parametrom pri návrhu bola možnosť horizontálneho škálovania, možnosť autorizácie a autentifikácie a zachovanie ako originálnych, nezmenených dát tak aj sledovanie prístupu a zmien v samotnom procese spracovania údajov.

Celá architektúra je založená na agilnom prístupe k tvorbe infraštruktúry a je možné ju prevádzkovať ako vo vlastnej firemnej infraštruktúre, tak aj v rámci cloudových riešení typu Amazon, Microsoft Azure, alebo Google Cloud Platform.

5.1 Spracovanie vstupných údajov

Ako sa ukázalo v kapitole 4, neexistuje jednotné riešenie pre spracovanie údajov, nakoľko sa dostávame do situácie, kde samotná réžia spojená s prácou v analytickom nástroji (Apache NiFi) môže zabrať dlhšiu dobu na spracovanie ako napísanie jednoduchého regulárneho výrazu, alebo python skriptu. Z toho dôvodu je navrhované rozdeliť spracovanie údajov na automatizované a poloautomatizované.

Poloautomatizované spracovanie dát prostredníctvom nástroja Apache NiFi

Medzi hlavné výhody nástroja Apache NiFi, patri možnosť dekompozície spracovania údajov na jednoduché procesory, black boxing a možnosť horizontálneho škálovania. Vďaka nástroju Apache

NiFi, je možné na spracovanie údajov využiť ľudské zdroje, ktoré nemusia mať znalosti v oblasti dátovej analytiky alebo programovania. Stačí ich zaškoliť do používania aplikácie Apache NiFi.

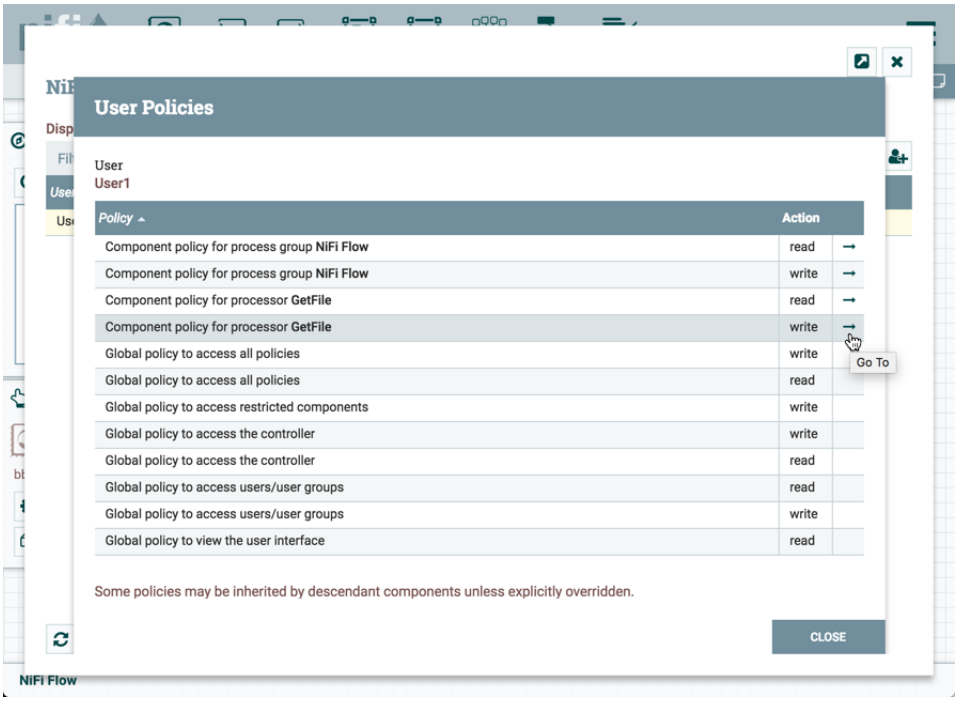
Zároveň Apache NiFi umožňuje združovanie do tzv. skupín, ktoré sa tvária ako čierne skrinky (black boxing), ktoré na vstupe očakávajú údaje a na výstupe doručia vyžadované informácie. V rámci možnosti aplikácie Apache NiFi je taktiež autentifikácia a autorizácia užívateľov, pričom autentifikáciu je možné prepojiť s centrálnym autentifikačným mechanizmom, nakoľko Apache NiFi podporuje nasledujúce spôsoby autentifikácie [28]:

- Lightweight Directory Access Protocol (LDAP)
- Kerberos
- OpenId Connect
- SAML
- Apache Knox

V rámci autorizácie je možné nastaviť obmedzenia na globálnej úrovni alebo na úrovni jednotlivých komponentov. Pre každú komponentu je možné nastaviť:

- Zobrazenie
- Modifikáciu

Správu užívateľských oprávnení je možné vidieť na obrázku 5.1. Automatizované riešenie je však výhodné hlavne pre súbory, ktoré sa dajú spracovať regulárnymi výrazmi a ktorých forma je takmer identická (wordlisty). Možný príklad takéhoto spracovania údajov, ktorý dokáže pracovať s údajmi z `Collection 1` a `CityODay`, je možné nájsť na obrázku 5.2.



Obr. 5.1: Záber zo správy užívateľských oprávnení v Apache NiFi [28]

Automatizované spracovanie údajov

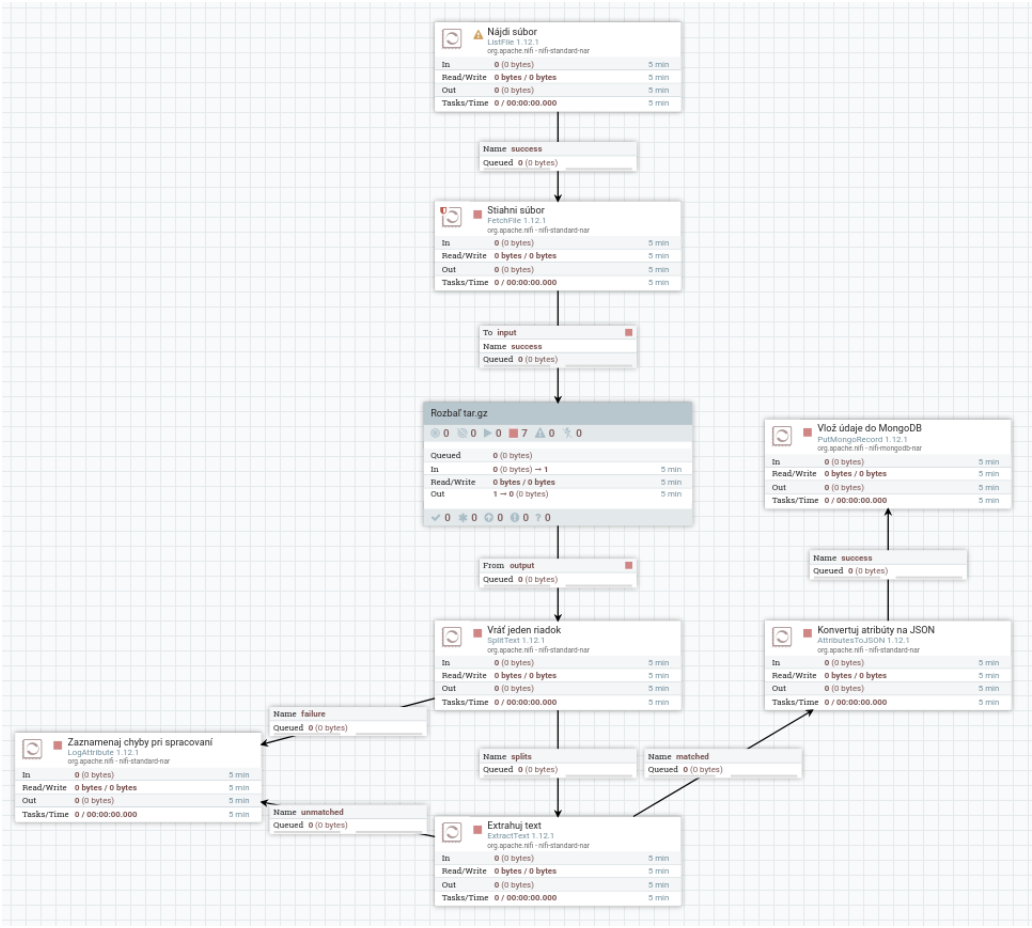
V rámci automatizovaného spracovania údajov bola vytvorená demo aplikácia v jazyku Python a framework-u Django. Táto aplikácia funguje na spôsob adaptéru, ktorý sa volá prostredníctvom príkazového riadku.

Zavolaním príkazu: `python manage.py import_data cesta_k_adresaru`, program automaticky nájde všetky archívy k dispozícii v danej zložke, rozbalí ich a následne hľadá súbor `normalization.py`, ktorý v sebe obsahuje generátor údajov a ktorý aplikácii poskytne potrebné údaje.

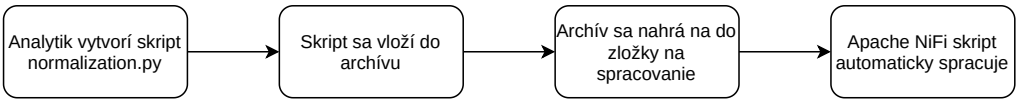
Aplikácia však bola navrhnutá hlavne za účelom testovania rýchlosti optimálneho riešenia pre zvolenú databázovú technológiu, demonštrovanie technológie a rýchlosti spracovania údajov v aplikácii Apache NiFi. Do výslednej architektúry sa odporúča nahradiť ju za vlastný procesor v aplikácii Apache NiFi.

Ostane len samotný skript na normalizáciu dát, ktorý je možné vidieť v prílohe D a spôsob spracovania, ktorý spočíva vo vložení pripraveného skriptu do archívu typu `tar.gz`. Výsledný postup v aplikácii Apache NiFi je zobrazený na obrázku 5.4[1] a postupnosť krokov, ktoré sa budú vyžadovať, je znázornená na grafe 5.3.

[1] Riešenie je iba názorné. Procesor, ktorý by dokázal pracovať s aplikáciu Apache NiFi aktuálne nie je vytvorený.



Obr. 5.2: Reálne spracovanie údajov prostredníctvom nástroja Apache NiFi [58]

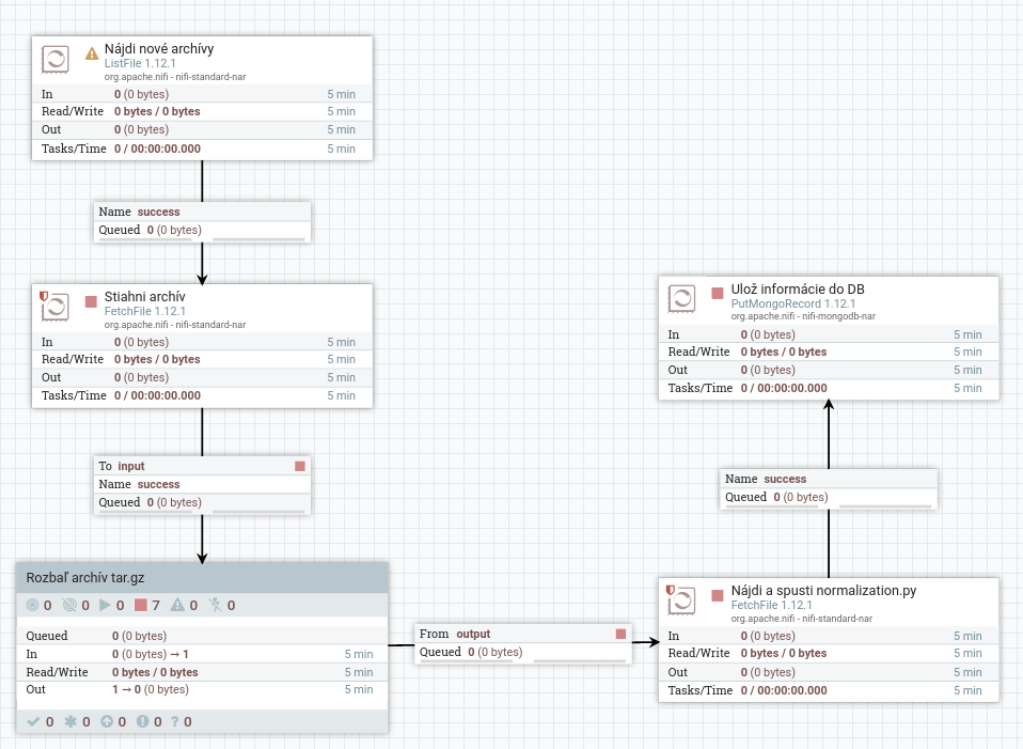


Obr. 5.3: Proces automatického spracovania údajov [58]

5.2 Použitá data storage technológia

Použitie vhodnej data storage technológie je dôležitým faktorom pri návrhu, nakoľko je potrebné zabezpečiť dostatočnú výkonnosť systému. Medzi dôležité parametre zároveň patrí aj podpora horizontálneho škálovania a možnosť zapnutia kompresie nad uloženými dátami.

Ako sa ukázalo, tak klasické relačné databázy štýlu PostgreSQL a MariaDB (MySQL) majú problémy so škálovaním nad jeden fyzický server a možnosti kompresie sú v prípade PostgreSQL značne obmedzené. Nakoľko medzi údajmi nie sú žiadne relácie, je možné opustiť segment riadkovo-orientovaných databáz (PostgreSQL) a prejsť do segmentu stĺpcovo orientovaných databáz, medzi ktoré patrí aj Greenplum. Z dôvodu čiastočnej kompatibility medzi PostgreSQL a



Obr. 5.4: Prípadný návrh automatizovaného spracovania v aplikácii Apache NiFi [58]

Greenplum je Greenplum vhodnou náhradou za PostgreSQL v oblasti Big data. Ako sa ukázalo v kapitole 4.5.1, Greenplum je možné škálovať horizontálne, je kompatibilný s nástrojmi z prostredia PostgreSQL a v porovnaní s klasickým PostgreSQL je taktiež rýchlejší, nakoľko údaje sa spracovávajú paralelne na viacerých uzloch.

Pozitívnou vlastnosťou je taktiež plná ACID kompatibilita. Medzi nevýhody však patrí nemožnosť upravovania existujúcich údajov. Jedinou možnosťou ako údaje upravovať, je získať daný záznam, uložiť si ho do medzi-pamäti, odstrániť originálny záznam a pridať upravený.

Problémom stĺpcovo orientovaných RDBMS je však potreba znalosti schémy pred samotným vkladaním údajov, čo je v prípade neštrukturovaných dát závažný problém. Riešením by bolo využiť dokument orientovanú NoSQL databázovú technológiu, akou je MongoDB. Tá síce patrí medzi najprimitívnejšie technológie z oblasti NoSQL, ale to na druhú stranu umožňuje jednoduchú implementáciu a prácu hlavne pre vývojárov a analytikov, ktorí stále úplne neopustili mentalitu relačných databázových systémov. Výhodou je taktiež podpora kompresie `bzip2`, ktorá umožňuje zmenšenie veľkosti archívu v niektorých prípadoch až o 70%.

Navrhované riešenie teda spočíva v ukladaní údajov do MongoDB databázového systému, ktorý je taktiež horizontálne škálovateľny a v prípade potreby je možné robiť exporty do relačnej databázy Greenplum prostredníctvom aplikácie Apache NiFi.

Na rozdiel od relačnej databázy MongoDB počíta s tým, že počet kolekcii (obdoba tabuliek v relačných databázach) môže dosiahnuť státisíce. V prípade relačných databáz by to však bolo považované sa chybný návrh. Ako je možné vidieť na obrázku 5.5, MongoDB bolo úspešne otestované vo vyhľadávaní údajov medzi 20 589 kolekciami, ktoré predstavujú detailnejšie rozdelenie údajov z `Collection 1` a `CityODay`. Na druhú stranu podobne veľký `JOIN` pre prehľadávanie v relačnej databáze by bol na hraniciach použiteľnosti.

Obr. 5.5: Záber na počet kolekcii v aplikácii Mongo Compass [58]

5.3 Front-end a overenie validity vstupných dát

Pre demonštračné účely a možnosti overenia spracovaných údajov bola vytvorená v jazyku Python a frameworku Django aplikácia, ktorá demonštruje výsledný stav, ktorého je možné dosiahnuť s použitím uvedených technológii.

Aplikácia je hybridom medzi aplikáciami haveibeenpwned.com a Intelligence X, ktoré boli popísané v sekcii 4.2.

Obr. 5.6: Základná stránka projektu je veľmi jednoduchá, zložená iba z vyhľadávacieho poľa [58]

Medzi jej funkcionalitu patrí:

- Prehľadávanie podľa presnej zhody
- Prehľadávanie podľa regulárneho výrazu
- Štatistiky prístupných údajov
- Autentifikácia a autorizácia

Aktuálne najrýchlejšie je vyhľadávanie podľa úplnej zhody, ktoré sa dá využiť napríklad na email alebo login. Informácie, ktoré sa zobrazia, reprezentujú záznam v MongoDB a predstavujú všetky informácie k danému záznamu.

Na druhu stranu veľmi pomalé je vyhľadávanie podľa regulárnych výrazov, kedy je možné vypísať si napríklad všetky záznamy pre špecifickú doménu. Ako je možné vidieť na obrázku 5.8, filtrovali sa záznamy pre doménu `vutbr.cz` a to prostredníctvom regulárneho výrazu `.*@vutbr\.cz`. Pri tomto spôsobe prehľadávania sa celá databáza prehľadáva sekvenčne a zrýchliť vyhľadávanie by tak bolo možné prostredníctvom horizontálneho škálovania, aby bol čo najväčší počet kolekcii rozprestretý cez čo najväčší počet fyzických zariadení. Jediným limitujúcim faktorom je v tomto prípade rýchlosť IO operácii.

Asi najdôležitejšou súčasťou sú štatistiky o údajoch v databáze, ktoré je možné vidieť na obrázku 5.9. V ľavom hornom rohu sú detailné informácie o počte kolekcii (tabuliek) k dispozícii, počte indexov, veľkosti dát (pred kompresiou), veľkosť, ktorú na disku zaberajú (po kompresii) a celkové voľné miesto na disku.

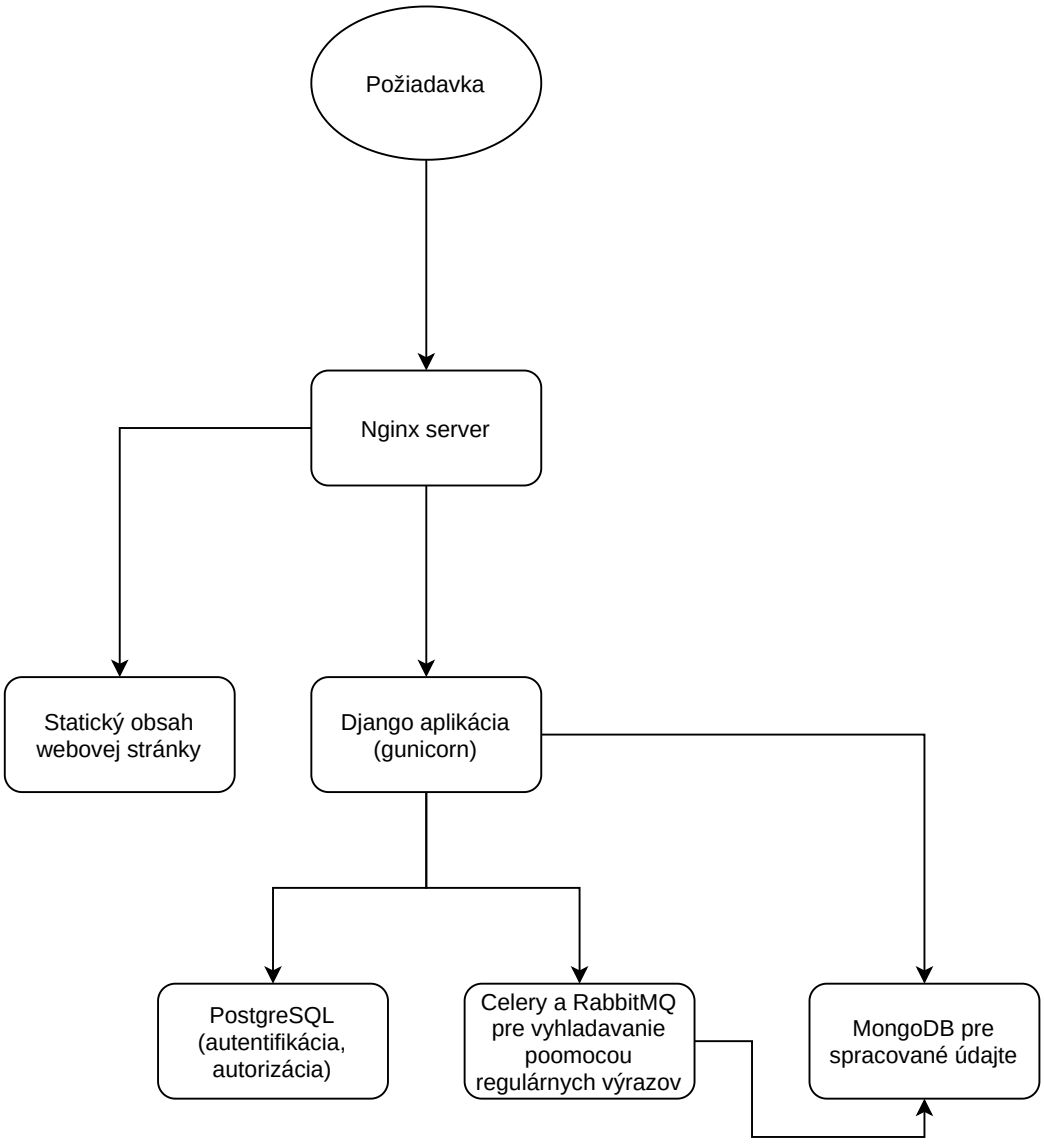
Čo sa týka počtu indexov, tak ten nikdy nebude menší, ako je počet kolekcii v databáze, nakoľko podobne ako pri relačných databázach, aj MongoDB indexuje primárny kľúč, ktorý je defaultne pomenovaný `_id` a ktorý musí obsahovať každý záznam[2]. `DB size total` predstavuje veľkosť, ktorú zaberajú údaje a index na disku spoločne.

Dôležitá je taktiež možnosť prehľadávania uložených údajov v rámci jednotlivých kolekcii, nakoľko je to rýchly a jednoduchý spôsob overenia si, že automatizované alebo poloautomatizované spracovanie údajov prebehlo v poriadku.

Pre produkčné nasadenie je ale odporúčané využiť namiesto vlastného riešenia aplikáciu Apache Solr, ktorá bola vytvorená práve za účelom rýchleho prehľadávania veľkého množstva údajov v rôznych formách. Na rozdiel od vlastného riešenia prichádza s možnosťami horizontálneho škálovania, replikáciou, širokou podporou importu údajov a REST API rozhraním na doprogramovanie vlastnej funkcionality.

V prípade nasadenia aktuálnej aplikácie je architektúra webovej aplikácie zobrazená na obrázku 5.7. Konfiguračné parametre sa nachádzajú v súbore `armeria_verimas/settings.py`.

[2]Čo však nie je pravda pre relačné databázy, nakoľko napríklad PostgreSQL umožňuje vytvorenie záznamu bez primárneho kľúča.



Obr. 5.7: Architektúra webovej aplikácie [58]

REDAMP SECURITY | Search by string | Search by regex | Stats | Sign up | Log in

email	password	path
mares@kn.vutbr.cz	spector	/media/firecracker/NVMe/tmp_data/romanohangos.cekit.cz (1.879) [HASH+NOHASH] (Business)_special_for_XSS.IS/Result.txt
mares@kn.vutbr.cz	402993a25bb45e2af21269a66ba74803	/media/firecracker/NVMe/tmp_data/romanohangos.cekit.cz (1.879) [HASH+NOHASH] (Business)_special_for_XSS.IS/romanohangos.cekit.cz (1.879) [HASH+NOHASH].txt
mares@kn.vutbr.cz	402993a25bb45e2af21269a66ba74803	/media/firecracker/NVMe/tmp_data/juniorbasket.cz (2.039) [HASH+NOHASH] (NoCategory)_special_for_XSS.IS/juniorbasket.cz (2.039) [HASH+NOHASH].txt
mares@kn.vutbr.cz	spector	/media/firecracker/NVMe/tmp_data/juniorbasket.cz (2.039) [HASH+NOHASH] (NoCategory)_special_for_XSS.IS/Result.txt
pouchly@fme.vutbr.cz	alicia	/media/firecracker/NVMe/tmp_data/thalie.pilsfree.cz (3.797) [HASH+NOHASH] (Business)_special_for_XSS.IS/Result.txt
pouchly@fme.vutbr.cz	*8FFB94ABC55619749163E336F440DAFD3F488F2C	/media/firecracker/NVMe/tmp_data/thalie.pilsfree.cz (3.797) [HASH+NOHASH] (Business)_special_for_XSS.IS/thalie.pilsfree.cz (3.797) [HASH].txt
191880@vutbr.cz	CCDVVj1j	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) final.txt
dosedelovap@study.fce.vutbr.cz	xarezka	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) final.txt
191880@vutbr.cz	CCDVVj1j	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) decrypted.txt
dosedelovap@study.fce.vutbr.cz	xarezka	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) decrypted.txt
191880@vutbr.cz	d31b3ebe25dd5c24c92922206a45ca2ff	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) [HASH] [NOHASH].txt
dosedelovap@study.fce.vutbr.cz	81271f77bed150f6012274a89df939e1	/media/firecracker/NVMe/tmp_data/seznam.czwww.mojenoty.cz (156.254) [HASH+NOHASH] (Entertainment)_special_for_XSS.IS/seznam.czwww.mojenoty.cz (156.254) [HASH] [NOHASH].txt
komarkova@fch.vutbr.cz	4a258d930b7d3409982d727ddbb4ba88	/media/firecracker/NVMe/tmp_data/pefka.net (25.564) [HASH+NOHASH] (Organizations&Government)_special_for_XSS.IS/www.pefka.net (25.564) [HASH].txt

Obr. 5.8: Regulárne výrazy (obdoba `LIKE`) sú síce pomalé, ale umožňujú vyhľadávanie napríklad podľa domény [58]

Obr. 5.9: Štatistiky majú prehľadné menu s uvedeným počtom spracovaných kolekcii (20 588) [58]

Obr. 5.10: Každá kolekcia sa dá prehliadnuť pre kontrolu spracovania [58]

5.4 Automatické overenie validity údajov v databáze

Aj napriek možnosti manuálnej kontroly je potrebné zabezpečiť možnosť kontrolovať konzistenciu a kvalitu dát automaticky. Dve základne možnosti, ako toho docieliť sú:

- Kontrola špecifických požiadaviek
- Štatistická kontrola
  - Tvorba histogramov
  - Počítanie regulačných diagramov

Medzi kontrolu špecifických požiadaviek by sa dali započítať veci, akým je napríklad maximálna dĺžka emailovej adresy, ktorá nemôže presiahnuť 320 znakov, nakoľko podľa štandardu RFC3696 [50]:

- Užívateľské meno môže mať maximálne 64 znakov
- Jeden znak pre symbol `@`
- Doména nemôže byť dlhšia ako 255 znakov

Podobne hodnoty je možné získať aj pre hash hesiel pre rôzne algoritmy. Ako príklad sa môže uviesť hashovací algoritmus SHA256, ktorý má presne 64 znakov, nakoľko hodnota je reprezentovaná hexadecimálnym zápisom (znaky v rozsahu 0 - 9 a A - F), pričom na zápis jedného znaku je potreba iba 4 bity, čo v prípade dĺžky 64 znakov znamená 256 bitov.

Nie v každom prípade je možné identifikovať presné parametre pre daný záznam a stĺpec. Niekedy je potrebné identifikovať chyby pomocou štatistických ukazovateľov, akými môže byť napríklad histogram priemernej dĺžky jednotlivých záznamov v databáze alebo teoreticky aj využitie regulačných diagramov na identifikovanie, či je daný proces (transformácia) v štatisticky zvládnutom stave.

Na tieto účely je však potrebné robiť rozsiahlu analýzu, ideálne nad celou množinou hodnôt v danej kolekcii, čo je náročné na zdroje a vo svojej podstate sa už jedná o analýzu Big data. K tomuto účelu budú využité nasledujúce technológie:

- Apache Spark
- Apache Drill

Každá z vyššie popísaných technológii má svoje silné a slabé stránky a aj napriek tomu, že na prvý pohľad ich účel môže vyzerať podobne, ich využitie je rozdielne.

Apache Spark poskytuje knižnicu PySpark pre programovanie v jazyku Python. To umožňuje jeho využitie na vytvorenie vlastných monitorovacích skriptov, zaisťujúcich kvalitu spracovaných dát a následne nasadenie monitorovania prostredníctvom aplikácie Apache Airflow. Medzi výhody taktiež patrí veľká komunita vývojárov, a teda aj veľká podpora na úrovni data storage systémov, vrátane Greenplum [22], MongoDB [25] a RDBMS systémy ako PostgreSQL a MariaDB prostredníctvom JDBC alebo ODBC. Dokonca má podporu spracovania údajov cez REST API a to prostredníctvom CData JDBC ovládača.

Medzi nevýhody patrí obmedzená podpora jazyka SQL, zatiaľ čo Apache Drill podporuje plnohodnotne štandard ANSI SQL 2003. Taktiež na rozdiel od Apache Drill, Apache Spark vyžaduje znalosť štruktúry údajov, ktoré spracováva. Umožňuje zároveň ako spracovanie toku dát, tak aj dátové spracovanie údajov. Okrem jazyka Python je možné využiť pre prístup aj nasledujúce jazyky:

- Java
- R
- Scala
- SQL

Celkovo by sa za výhody Apache Spark dalo považovať:

- Široká podpora zo strany storage enginov
- Široký výber programovacích jazykov na prístup
- Podpora stream processingu a dávkového spracovania
- 100x rýchlejší ako MapReduce v pamäti a 10x rýchlejší na disku

Medzi jeho hlavne nevýhody patrí:

- Bez podpory real time processingu
- Obmedzená podpora SQL
- Nutná znalosť štruktúry údajov

Apache Drill má na rozdiel od Apache Spark plnohodnotnú podporu štandardu ANSI SQL 2003, čo je výhodné hlavne z dôvodu širokej znalosti jazyka SQL v oblasti dátovej analytiky a možnosti spustenia komplexných agregačných dotazov. Taktiež pre svoje fungovanie nevyžaduje znalosť štruktúry údajov a má širokú podporu BI nástrojov, medzi ktoré patria:

- Tableau
- Qlik
- MicroStrategy
- Spotfire
- SAS
- Micorosoft Excel

Apache Drill podporuje real-time spracovanie údajov a má dokonca väčšiu podporu data storage enginov ako Apache Spark, nakoľko pre svoje fungovanie nevyžaduje znalosť štruktúry údajov. Na rozdiel od Apache Spark však nemá podporu v ďalších jazykoch a prístup k údajom je možný len v podobe SQL príkazov. Taktiež má menšiu komunitu ako Apache Spark.

Zaujímavé je však ich spoločné využitie [5], nakoľko Apache Drill podporuje JDBC ovládač a Apache Spark vie použiť JDBC ako zdroj dát. To znamená, že je možné použiť Apache Spark na transformáciu dát, zatiaľ čo samotné dáta sa získajú prostredníctvom Apache Drill do Apache Spark.

5.5 Návrh finálnej architektúry

Výsledná architektúra, ktorú je možné vidieť na obrázku 5.11, bola navrhnutá v duchu metodiky DevOps, berúc do úvahy nasledujúce vlastnosti:

- Škálovateľnosť
- Modulárnosť
- Dynamickosť

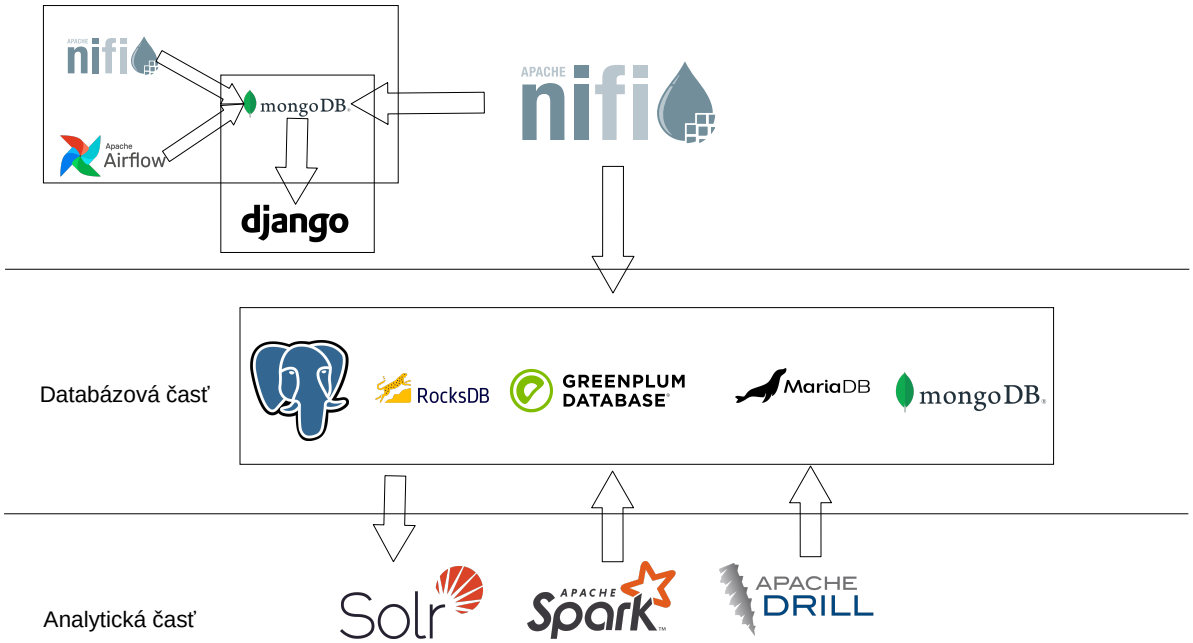
Každá jedná komponenta v architektúre môže byť zameniteľná. Toho sa dosahuje hlavne s využitím nástroja Apache NiFi, ktorý umožňuje prostredníctvom svojho rozhrania stiahnuť, transformovať a nahrať ľubovolný typ údajov do rôznorodej databázy. V prípade tvorby exportov, pri ktorých sa očakáva čiastočné orezanie alebo anonymizovanie údajov, je vhodné využiť aplikáciu Apache Spark, ktorá poskytuje knižnice na prácu v jazyku Python a ktorá je priamo určená na spracovanie Big data.

Apache Spark a Apache Drill sa zároveň môžu využiť na analytické účely, vrátane manažmentu kvality uložených údajov, pričom Apache Drill poskytuje kompletnú implementáciu SQL štandardu ANSI SQL 2003 a využíva jazyk SQL ako primárny nástroj pre prístup k údajom uložených v ľubovolnom type ako relačnej, tak aj NoSQL databáze. V prípade potreby je možné využiť kombináciu Apache Spark pre spracovanie a Apache Drill pre prístup k údajom a to v prípade, že by sa údaje ukladali na viac ako jeden typ média. Príkladom by mohlo byť využitie technológie HDFS, ktorá však pre aktuálny prípad užitia nemá opodstatnenie.

V prípade potreby, nahradiť vytvorenú aplikáciu v jazyku Django, sa odporúča zvoliť Apache Solr, ktorý je navrhnutý už priamo ako vyhľadávací engine nad Big data a ktorý rieši mnohé problémy, ktoré by bolo treba znovuobjaviť a vyriešiť v rámci vývoja aplikácie na zelenej lúke.

Na spracovanie neštrukturovaných vstupných dát sa odporúča využiť aplikáciu Apache NiFi, s prípadným doprogramovaním modulu, ktorý by umožnil načítanie údajov prostredníctvom skriptu uvedeného v prílohe D a ktorý aktuálne je využívaný aplikáciu v jazyku Django na demonštrovanie funkcionality poloautomatického spracovania údajov.

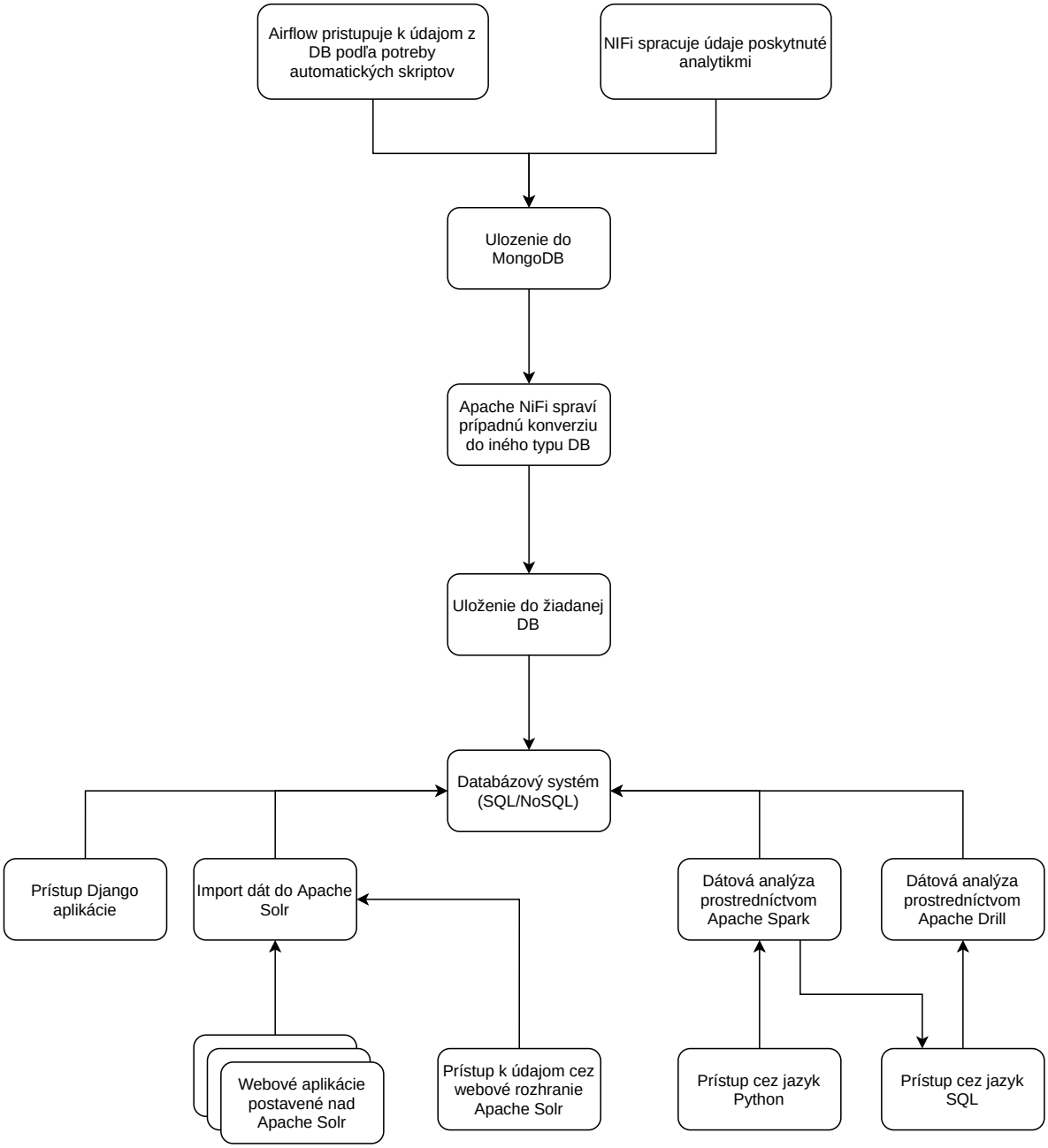
Pre uloženie údajov sa odporúča využitie databázového systému MongoDB, ktorý je jednoduchý na použitie, horizontálne škálovateľný a aj kvôli svojej jednoduchosti odporúčaný do začiatku. Výhodou je taktiež možný prístup zo strany front-endu, aj keď to nie je odporúčané z bezpečnostných dôvodov.



Obr. 5.11: Návrh finálnej architektúry pre efektívnu analýzu dát [58]

Apache Airflow je odporúčaný na spúšťanie periodických skriptov, ktoré môžu mať podpornú alebo rozširujúcu funkcionalitu v architektúre. Ako možný príklad ich využitia je tvorba programu, ktorý upozorní danú spoločnosť v prípade objavenia sa nových údajov v databázy pre danú doménu. Pre tento účel je potreba využiť regulárne výrazy, čo je drahá a pomalá operácia. Zároveň je to operácia, ktorá sa musí robiť periodicky (napríklad jedenkrát za deň).

V prípade rozhodnutia použiť klasickú relačnú databázu, odporúča sa zvolenie technológie Greenplum, ktorá podporuje kompresiu na úrovni stĺpcov a zároveň je kompatibilná so syntaxou starších verzii PostgreSQL (8 a 9).



Obr. 5.12: Proces fungovania finálnej architektúry [58]

Kapitola 6

Záver

V analytickej časti boli popísané možné spôsoby spracovania veľkého objemu neštrukturovaných dát, nakoľko nie je možné takýto objem dát spracovať manuálne. Taktiež boli popísané existujúce riešenia a spísané dôvody, prečo je analýzu a zber takých údajov potreba robiť.

Zároveň bolo zanalyzovaných viac ako 2TB dát, ktoré budu slúžiť ako vstupné, nespracované údaje pre výsledný návrh architektúry. Bolo popísané zloženie súborov podľa formátu, veľkosti, a ich pomeru. Boli popísané problémy, ktoré bolo potreba vyriešiť pred samotným spracovaním. Za týmto účelom boli detailne spracované kolekcie `Collection 1` a `City0Day`, ktoré spoločne obsahovali viac ako 3,7 miliardy údajov. Táto sada informácii slúžila ako vstupné dáta pre porovnanie parametrov troch úložných technológii, ktorými sú Greenplum, PostgreSQL a MongoDB. V záťažových testoch sa porovnávalo miesto, ktoré zaberajú údaje na disku a zároveň výkon pri spracovaní údajov.

Čo sa týka úložného miesta, tak najhoršie vyšla technológia PostgreSQL, pri ktorej samotné dáta zaberali 593 GB oproti technológii Greenplum, kde rovnaký objem a zloženie dát zaberal len nepatrných 57 GB priestoru. V otázke rýchlosti sa porovnávala hlavne technológia PostgreSQL a MongoDB a v tomto porovnaní o niečo lepšie viedla technológia MongoDB. Zo spomenutých technológii horizontálne škálovanie nepodporuje iba PostgreSQL.

Na základe týchto údajov sa pristúpilo k samotnému návrhu výslednej infraštruktúry. Boli popísané a navrhnuté spôsoby, ako dáta spracovať automatizovane a poloautomatizovane. Boli popísané výhody ako jedného, tak aj druhého spôsobu riešenia. Bola popísaná zvolená databázová technológia (MongoDB) a dôvody jej voľby. Na demonštráciu možnosti bola následne vytvorená jednoduchá webová aplikácia, ktorá dokázala dáta prečítať a prehľadávať, alebo si zobraziť informácie o prístupných údajoch.

Taktiež boli popísané spôsoby, ako je možné v danej architektúre overovať kvalitu uložených údajov a ako výsledné dáta sprístupniť k analytickým alebo BI účelom a to ideálne vo forme prístupu cez jazk SQL alebo ako zdroj pre najrozšírenejšie BI nástroje.

Výsledná architektúra je popísaná v poslednej časti a celý návrh je veľmi agilný a postavaný formou blokov, s možnosťami zmien v ktorejkoľvek časti. Vďaka vhodnému návrhu nie je ani databázový systém fixný a je možné ho zameniť za ľubovolný iný systém.

Všetky technológie použité pri návrhu sú open-source a za ich používanie nie je potrebné platiť žiadne licenčné poplatky. Z toho dôvodu bola kapitola finančných nákladov vynechaná. Firma, pre ktorú bol návrh infraštruktúry vytvorený, má svoju vlastnú fyzickú infraštruktúru a teda sa neočakávajú ani kapitálove výdavky na fyzickú infraštruktúru.

Aj napriek tomu, že architektúra bude prevádzkovaná na vlastnej infraštruktúre, návrh počíta aj s možným prechodom do cloud-u a celý systém je tak možné nasadiť aj bez vlastnej infraštruktúry.

Literatúra

[1] 2021 Credential Stuffing Report. [online]. [cit. 2021-02-13]. Dostupné z: https://www.f5.com/labs/articles/threat-intelligence/2021-credential-stuffing-report.

[2] 52% of users reuse their passwords. [online]. [cit. 2021-05-13]. Dostupné z: https://www.pandasecurity.com/en/mediacenter/security/password-reuse/.

[3] The 773 Million Record "Collection #1"Data Breach. [online]. [cit. 2021-05-13]. Dostupné z: https://www.troyhunt.com/the-773-million-record-collection-1-data-reach/.

[4] *About MyRocks for MariaDB* [online]. [cit. 2021-05-13]. Dostupné z: https://mariadb.com/kb/en/about-myrocks-for-mariadb/.

[5] Apache Drill vs. Apache Spark — Which SQL query engine is better for you? [online]. [cit. 2021-02-05]. Dostupné z: https://towardsdatascience.com/apache-drill-vs-apache-spark-which-sql-query-engine-is-better-for-you-2a43f381bcd7.

[6] *Apache Hadoop* [online]. [cit. 2021-01-10]. Dostupné z: https://hadoop.apache.org/.

[7] *Apache NiFi Overview* [online]. [cit. 2021-01-10]. Dostupné z: https://nifi.apache.org/docs/nifi-docs/html/overview.html.

[8] *Apache Spark FAQ* [online]. [cit. 2021-01-10]. Dostupné z: https://spark.apache.org/faq.html.

[9] *Cassandra Basics* [online]. [cit. 2021-05-13]. Dostupné z: https://cassandra.apache.org/cassandra-basics/.

[10] Clubhouse data leak: 1.3 million scraped user records leaked online for free. [online]. [cit. 2021-05-13]. Dostupné z: https://cybernews.com/security/clubhouse-data-leak-1-3-million-user-records-leaked-for-free-online/.

[11] *ConsumeAzureEventHub - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-azure-nar/1.13.2/org.apache.nifi.processors.azure.eventhub.ConsumeAzureEventHub/index.html.

[12] *ConvertAvroToJSON - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-avro-nar/1.13.2/org.apache.nifi.processors.avro.ConvertAvroToJSON/index.html.

[13] *Documentation of PostgreSQL 13: 19.4. Resource Consumption* [online]. [cit. 2021-02-21]. Dostupné z: https://www.postgresql.org/docs/13/runtime-config-resource.html.

[14] *ExtractText - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.13.2/org.apache.nifi.processors.standard.ExtractText/index.html.

[15] Facebook says a breach that hit 533m is old news. Experts disagree. [online]. [cit. 2021-05-13]. Dostupné z: https://www.theguardian.com/technology/2021/apr/11/another-huge-data-breach-another-stony-silence-from-facebook.

[16] *GetFTP - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.13.2/org.apache.nifi.processors.standard.GetFTP/index.html.

[17] *GetMongoRecord - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-mongodb-nar/1.13.2/org.apache.nifi.processors.mongodb.GetMongoRecord/index.html.

[18] *GetTwitter - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-social-media-nar/1.13.2/org.apache.nifi.processors.twitter.GetTwitter/index.html.

[19] *Greenplum Database Concepts: About the Greenplum Architecture* [online]. [cit. 2021-05-13]. Dostupné z: https://gpdb.docs.pivotal.io/5160/admin_guide/intro/arch_overview.html.

[20] Hacker leaks passwords for 900+ enterprise VPN servers. [online]. [cit. 2021-05-11]. Dostupné z: https://www.zdnet.com/article/hacker-leaks-passwords-for-900-enterprise-vpn-servers/.

[21] *Intelligence X is a search engine and data archive.* [online]. [cit. 2021-05-13]. Dostupné z: https://intelx.io/about.

[22] Introducing Pivotal Greenplum-Spark Connector, Integrating with Apache Spark. [online]. [cit. 2021-05-13]. Dostupné z: https://greenplum.org/introducing-pivotal-greenplum-spark-connector-integrating-apache-spark/.

[23] *InvokeAWSGatewayApi - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-aws-nar/1.13.2/org.apache.nifi.processors.aws.wag.InvokeAWSGatewayApi/index.html.

[24] MobiKwik: Message from the company. [online]. [cit. 2021-05-13]. Dostupné z: https://blog.mobikwik.com/message-from-the-company/.

[25] *MongoDB Connector for Apache Spark* [online]. [cit. 2021-05-13]. Dostupné z: https://www.mongodb.com/products/spark-connector.

[26] *MongoDB The application data platform* [online]. [cit. 2021-05-13]. Dostupné z: https://www.mongodb.com.

[27] *MyRocks Documentation* [online]. [cit. 2021-05-13]. Dostupné z: http://myrocks.io/docs/getting-started/.

[28] NiFi System Administrator's Guide. [online]. [cit. 2021-01-27]. Dostupné z: https://nifi.apache.org/docs/nifi-docs/html/administration-guide.html.

[29] NSA 'NiFi' Big Data Automation Project Out In The Open. [online]. [cit. 2021-01-10]. Dostupné z: https://www.forbes.com/sites/adrianbridgwater/2015/07/21/nsa-nifi-big-data-automation-project-out-in-the-open.

[30] *Official webpage of project Apache Airflow* [online]. [cit. 2021-05-11]. Dostupné z: https://airflow.apache.org/.

[31] *Official webpage of project Apache Drill* [online]. [cit. 2021-05-11]. Dostupné z: https://drill.apache.org.

[32] *PostgreSQL: About* [online]. [cit. 2021-05-13]. Dostupné z: https://www.postgresql.org/about/.

[33] PostgreSQL: effective_io_concurrency benchmarked. [online]. [cit. 2021-02-21]. Dostupné z: https://portavita.github.io/2019-07-19-PostgreSQL_effective_io_concurrency_benchmarked/.

[34] *PutEmail - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.13.2/org.apache.nifi.processors.standard.PutEmail/index.html.

[35] *PutSQL - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.13.2/org.apache.nifi.processors.standard.PutSQL/index.html.

[36] Scraped data of 500 million LinkedIn users being sold online, 2 million records leaked as proof. [online]. [cit. 2021-05-13]. Dostupné z: https://cybernews.com/news/stolen-data-of-500-million-linkedin-users-being-sold-online-2-million-leaked-as-proof-2/.

[37] *SplitText - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-standard-nar/1.13.2/org.apache.nifi.processors.standard.SplitText/index.html.

[38] THE NETFLIX TECH BLOG: Hadoop Platform as a Service in the Cloud. [online]. [cit. 2021-05-11]. Dostupné z: https://netflixtechblog.com/hadoop-platform-as-a-service-in-the-cloud-c23f35f965e7.

[39] *Trademark Guidelines* [online]. [cit. 2021-05-13]. Dostupné z: https://greenplum.org/trademark/.

[40] What is Apache Solr? And why you should use it. [online]. [cit. 2021-05-11]. Dostupné z: https://www.infoworld.com/article/3209685/why-you-should-use-apache-solr.html.

[41] *YandexTranslate - Apache NiFi* [online]. [cit. 2021-01-10]. Dostupné z: http://nifi.apache.org/docs/nifi-docs/components/org.apache.nifi/nifi-language-translation-nar/1.13.2/org.apache.nifi.processors.yandex.YandexTranslate/index.html.

[42] Zedstore – Compressed Columnar Storage for Postgres. [online]. [cit. 2021-05-13]. Dostupné z: https://blogs.vmware.com/opensource/2020/07/14/zedstore-compressed-columnar-storage-for-postgres/.

[43] *The "Have I been pwned?" Microsoft Azure Ecosystem* [online]. [cit. 2021-05-13]. Dostupné z: https://haveibeenpwned.com/ecosystem.pdf.

[44] *XRDS*. Association for Computing Machinery. 2012, zv. 19, č. 1. ISSN 1528-4972.

[45] Configuring and Tuning PostgreSQL and EDB Postgres Advanced Server - Guide for Linux Users. [online]. EnterpriseDB. 2020. Dostupné z: https://info.enterprisedb.com/whitepaper_PostgreSQL-and-EPAS-Guide-Linux.html.

[46] BENITA, H. Fastest Way to Load Data Into PostgreSQL Using Python. [online]. 2019, [cit. 2021-03-13]. Dostupné z: https://hakibenita.com/fast-load-data-python-postgresql.

[47] CAVANILLAS, J., CURRY, E. a WAHLSTER, W. *New Horizons for a Data-Driven Economy: A Roadmap for Usage and Exploitation of Big Data in Europe*. Január 2015. ISBN 978-3-319-21569-3.

[48] KARCHER, T. a LANDHÄUSSER, M. Poster: Autotuning PostgreSQL: A Blueprint for Successful Autotuning of Real-World Applications. In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. 2018, s. 436–437.

[49] KHAILANY, B., WILLIAMS, T., LIN, J., LONG, E., RYGH, M. et al. A Programmable 512 GOPS Stream Processor for Signal, Image, and Video Processing. In: *2007 IEEE International Solid-State Circuits Conference. Digest of Technical Papers*. 2007, s. 272–602. DOI: 10.1109/ISSCC.2007.373399.

[50] KLENSIN, D. J. C. *Application Techniques for Checking and Transformation of Names* [RFC 3696]. RFC Editor, február 2004. DOI: 10.17487/RFC3696. Dostupné z: https://rfc-editor.org/rfc/rfc3696.txt.

[51] LABERGE, R. *The Data Warehouse Mentor: Practical Data Warehouse and Business Intelligence Insights*. 1st. McGraw-Hill Education Group, 2011. ISBN 0071745327.

[52] LINNAKANGAS, H. *ZedStore – Column store for PostgreSQL* [online]. [cit. 2021-03-04]. Dostupné z: https://www.postgresql.eu/events/pgconfeu2019/sessions/session/2738/slides/233/ZedStore-PGConfeu2019-Milan.pdf.

[53] LU, H., NG, Y. Y. a TIAN, Z. T-tree or B-tree: main memory database index structure revisited. In: *Proceedings 11th Australasian Database Conference. ADC 2000 (Cat. No.PR00528)*. 2000, s. 65–73. DOI: 10.1109/ADC.2000.819815.

[54] MATSUNOBU, Y., DONG, S. a LEE, H. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. *Proc. VLDB Endow.* VLDB Endowment. august 2020, zv. 13, č. 12, s. 3217–3230. DOI: 10.14778/3415478.3415546. ISSN 2150-8097. Dostupné z: https://doi.org/10.14778/3415478.3415546.

[55] MEIER, A. a KAUFMANN, M. *SQL & NoSQL Databases*. Wiesbaden: Springer Fachmedien Wiesbaden, 2019. ISBN 978-3-658-24548-1.

[56] PATTERSON, D. A., GIBSON, G. a KATZ, R. H. A Case for Redundant Arrays of Inexpensive Disks (RAID). *SIGMOD Rec.* New York, NY, USA: Association for Computing Machinery. jún 1988, zv. 17, č. 3, s. 109–116. DOI: 10.1145/971701.50214. ISSN 0163-5808. Dostupné z: https://doi.org/10.1145/971701.50214.

[57] PLASE, D., NIEDRITE, L. a TARANOVS, R. Accelerating data queries on Hadoop framework by using compact data formats. In: *2016 IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2016, s. 1–7. DOI: 10.1109/AIEEE.2016.7821807.

[58] PODLESNÝ Šimon. Vlastné spracovanie.

[59] SAKR, S. a ZOMAYA, A. Y., ed. *Encyclopedia of Big Data Technologies*. Springer, 2019. ISBN 978-3-319-77526-5. Dostupné z: https://doi.org/10.1007/978-3-319-77526-5.

[60] SARAN, N. Pandas to PostgreSQL using Psycopg2: Bulk Insert Performance Benchmark. [online]. 2020, [cit. 2021-02-21]. Dostupné z: https://naysan.ca/2020/05/09/pandas-to-postgresql-using-psycopg2-bulk-insert-performance-benchmark/.

[61] SHARDA, R., DELEN, D. a TURBAN, E. *Business Intelligence: A Managerial Perspective on Analytics (3rd Edition)*. 4rd. USA: Pearson, 2017. ISBN 78-1-292-22054-3.

[62] THOMAS, S. *PostgreSQL 12 High Availability Cookbook: Over 100 Recipes to Design a Highly Available Server with the Advanced Features of PostgreSQL 12, 3rd Edition*. Packt Publishing, Limited, 2020. ISBN 1838984852.

[63] WORSLEY, J. C. a DRAKE, J. D. *Practical PostgreSQL*. OReilly, 2002. ISBN 9781565928466.

Prílohy

Zoznam príloh

Príloha A

Záťažový test pre databázu PostgreSQL

```
1  select extract(epoch from now()) AS timer;
2
3  DROP INDEX IF EXISTS email_hash_city0day;
4  select extract(epoch from now()) AS timer;
5
6  DROP INDEX IF EXISTS email_hash_collection1;
7  select extract(epoch from now()) AS timer;
8
9  -- SELECT COUNT(*) FROM city0day;
10 SELECT COUNT(email) FROM city0day;
11 select extract(epoch from now()) AS timer;
12
13 -- SELECT COUNT(*) FROM collection1;
14 SELECT COUNT(email) FROM collection1;
15 select extract(epoch from now()) AS timer;
16
17 -- SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪  city0day.email=collection1.email WHERE collection1.email =
   ↪  'spodlesny@gmail.com';
18 SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪  city0day.email=collection1.email WHERE collection1.email =
   ↪  'spodlesny@gmail.com';
```

```
19 select extract(epoch from now()) AS timer;
20
21 -- SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
22 SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
23 select extract(epoch from now()) AS timer;
24
25 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  cityOday;
26 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  cityOday;
27 select extract(epoch from now()) AS timer;
28
29 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  collection1;
30 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  collection1;
31 select extract(epoch from now()) AS timer;
32
33 -- CREATE INDEXES
34 CREATE INDEX CONCURRENTLY email_hash_collection1 ON collection1 (email);
35 CREATE INDEX CONCURRENTLY email_hash_cityOday on cityOday (email);
36
37 select extract(epoch from now()) AS timer;
38
39 -- SELECT COUNT(*) FROM cityOday;
40 SELECT COUNT(email) FROM cityOday;
41 select extract(epoch from now()) AS timer;
42
43 -- SELECT COUNT(*) FROM collection1;
44 SELECT COUNT(email) FROM collection1;
45 select extract(epoch from now()) AS timer;
46
```

```
47 -- SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪ city0day.email=collection1.email WHERE collection1.email =
   ↪ 'spodlesny@gmail.com';
48 SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪ city0day.email=collection1.email WHERE collection1.email =
   ↪ 'spodlesny@gmail.com';
49 select extract(epoch from now()) AS timer;
50
51 -- SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
52 SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
53 select extract(epoch from now()) AS timer;
54
55 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ city0day;
56 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ city0day;
57 select extract(epoch from now()) AS timer;
58
59 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ collection1;
60 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ collection1;
61 select extract(epoch from now()) AS timer;
```

Príloha B

Záťažový test pre databázu Greenplum

```
1  select extract(epoch from now()) AS timer;
2
3  DROP INDEX IF EXISTS email_hash_city0day;
4  select extract(epoch from now()) AS timer;
5
6  DROP INDEX IF EXISTS email_hash_collection1;
7  select extract(epoch from now()) AS timer;
8
9  -- SELECT COUNT(*) FROM city0day;
10 SELECT COUNT(email) FROM city0day;
11 select extract(epoch from now()) AS timer;
12
13 -- SELECT COUNT(*) FROM collection1;
14 SELECT COUNT(email) FROM collection1;
15 select extract(epoch from now()) AS timer;
16
17 -- SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪  city0day.email=collection1.email WHERE collection1.email =
   ↪  'spodlesny@gmail.com';
18 SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪  city0day.email=collection1.email WHERE collection1.email =
   ↪  'spodlesny@gmail.com';
```

```
19 select extract(epoch from now()) AS timer;
20
21 -- SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
22 SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
23 select extract(epoch from now()) AS timer;
24
25 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  city0day;
26 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  city0day;
27 select extract(epoch from now()) AS timer;
28
29 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  collection1;
30 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪  collection1;
31 select extract(epoch from now()) AS timer;
32
33 -- CREATE INDEXES
34 CREATE INDEX email_hash_collection1 ON collection1(email);
35 CREATE INDEX email_hash_city0day on city0day(email);
36
37 select extract(epoch from now()) AS timer;
38
39 -- SELECT COUNT(*) FROM city0day;
40 SELECT COUNT(email) FROM city0day;
41 select extract(epoch from now()) AS timer;
42
43 -- SELECT COUNT(*) FROM collection1;
44 SELECT COUNT(email) FROM collection1;
45 select extract(epoch from now()) AS timer;
46
```

```
47 -- SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪ city0day.email=collection1.email WHERE collection1.email =
   ↪ 'spodlesny@gmail.com';
48 SELECT COUNT(*) FROM city0day INNER JOIN collection1 ON
   ↪ city0day.email=collection1.email WHERE collection1.email =
   ↪ 'spodlesny@gmail.com';
49 select extract(epoch from now()) AS timer;
50
51 -- SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
52 SELECT COUNT(*) FROM collection1 WHERE email LIKE '%spodlesny%';
53 select extract(epoch from now()) AS timer;
54
55 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ city0day;
56 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ city0day;
57 select extract(epoch from now()) AS timer;
58
59 -- SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ collection1;
60 SELECT COUNT(DISTINCT filename) as "Number of processed files" FROM
   ↪ collection1;
61 select extract(epoch from now()) AS timer;
```

Príloha C

Záťažový test pre databázu MongoDB

```
#! /bin/bash
cl1="City0Day"

echo -e "\n Drop indexes if exist: "
time mongo --eval 'db["City0Day"].dropIndex({"email": "hashed"})'
↪ armeria_verimas
time mongo --eval 'db["Collection 1"].dropIndex({"email": "hashed"})'
↪ armeria_verimas

echo "Measuring count() time: "
time mongo --eval 'db["City0Day"].count()' armeria_verimas
time mongo --eval 'db["Collection 1"].count()' armeria_verimas

echo -e "\n Measuring alternative to SELECT: "
time mongo --eval 'db["City0Day"].find({"email":"spodlesny@gmail.com"})'
↪ armeria_verimas
time mongo --eval 'db["Collection
↪ 1"].find({"email":"spodlesny@gmail.com"})' armeria_verimas

echo -e "\n Measuring alternative to LIKE: "
time mongo --eval 'db["Collection 1"].find({"email":/spodlesny/}).count()'
↪ armeria_verimas
```

```
18
19 echo -e "\n Measuring DISTINCT: "
20 time mongo --eval 'db["City0Day"].distinct("filename").count()'
   ↪ armeria_verimas
21 time mongo --eval 'db["Collection 1"].distinct("filename").count()'
   ↪ armeria_verimas
22
23 echo -e "\n Create indexes: "
24 time mongo --eval 'db["City0Day"].createIndex({"email": "hashed"})'
   ↪ armeria_verimas
25 time mongo --eval 'db["Collection 1"].createIndex({"email": "hashed"})'
   ↪ armeria_verimas
26
27 echo "Measuring count() time: "
28 time mongo --eval 'db["City0Day"].count()' armeria_verimas
29 time mongo --eval 'db["Collection 1"].count()' armeria_verimas
30
31 echo -e "\n Measuring alternative to SELECT: "
32 time mongo --eval 'db["City0Day"].find({"email":"spodlesny@gmail.com"})'
   ↪ armeria_verimas
33 time mongo --eval 'db["Collection
   ↪ 1"].find({"email":"spodlesny@gmail.com"})' armeria_verimas
34
35 echo -e "\n Measuring alternative to LIKE: "
36 time mongo --eval 'db["Collection 1"].find({"email":/spodlesny/}).count()'
   ↪ armeria_verimas
37
38 echo -e "\n Measuring DISTINCT: "
39 time mongo --eval 'db["City0Day"].distinct("filename").count()'
   ↪ armeria_verimas
40 time mongo --eval 'db["Collection 1"].distinct("filename").count()'
   ↪ armeria_verimas
```

Príloha D

Generátor údajov v jazyku Python

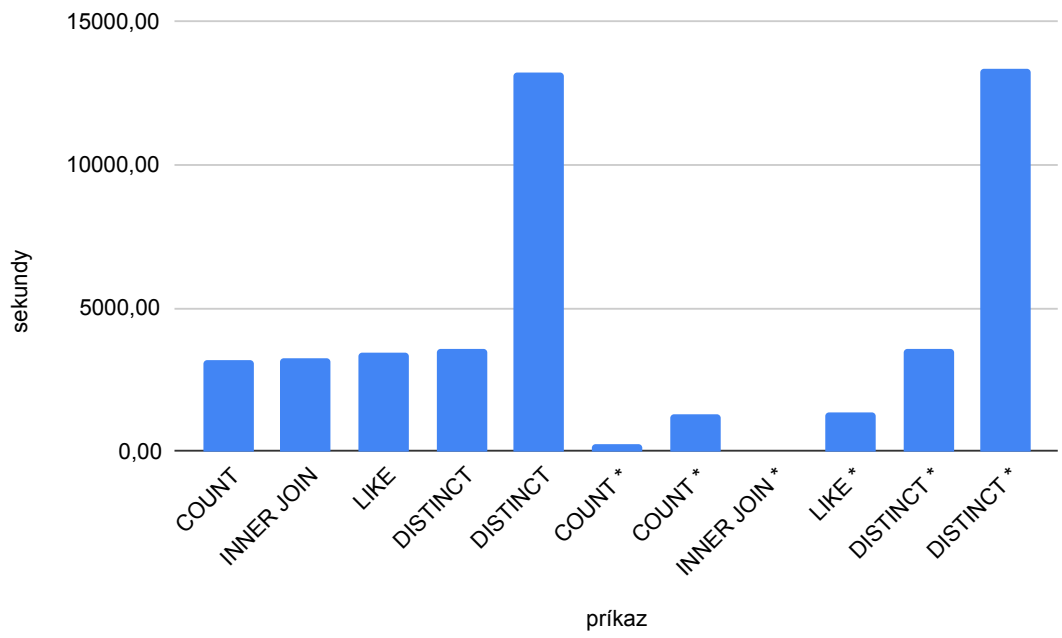
```
import glob
import logging
import os

logger = logging.getLogger(__name__)


path = os.path.abspath(__file__)
dir_path = os.path.dirname(path)
os.chdir(dir_path)
list_of_files = glob.glob("*.txt")

source_name = dir_path.split("/")[-1]


def normalize_data(data):
    email, password = "", ""
    try:
        email, password = data.split(":", 1)
    except ValueError:
        try:
            email, password = data.split(";", 1)
```

```
            except Exception as err:
                logger.debug(f"Line '{data}' cannot be processed. Reason: {err}")
        except Exception as err:
            # unhandled error
            logger.error(f"Unhandled exception `{err}` for data: {data}.")
            exit(1)

        return email, password


def get_records():
    for filename in list_of_files:
        filename = dir_path + "/" + filename
        with open(filename, "r", encoding="utf-8", errors="ignore") as file:
            for line in file:
                line = line.strip().replace(
                    "\x00", ""
                )
                if line == "" or len(line) == 1:
                    continue
                email, password = normalize_data(line)
                yield {
                    "source_name": source_name,
                    "email": email,
                    "password": password,
                    "file_name": filename,
                }


if __name__ == "__main__":
    generator = get_records()
    print(next(generator))
```

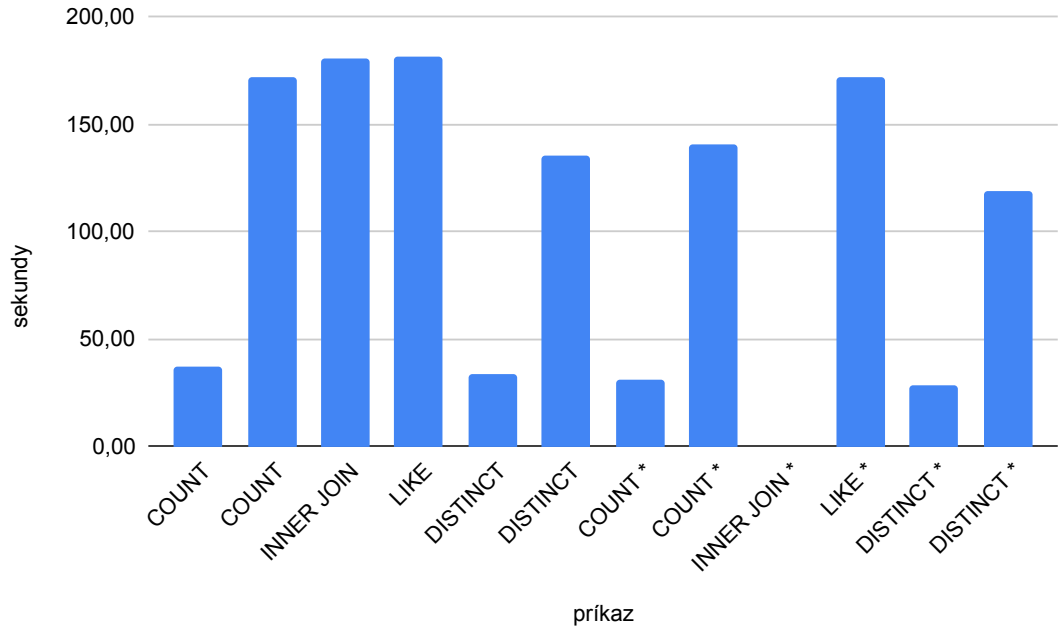
```
print(next(generator))
print(next(generator))
```

Príloha E

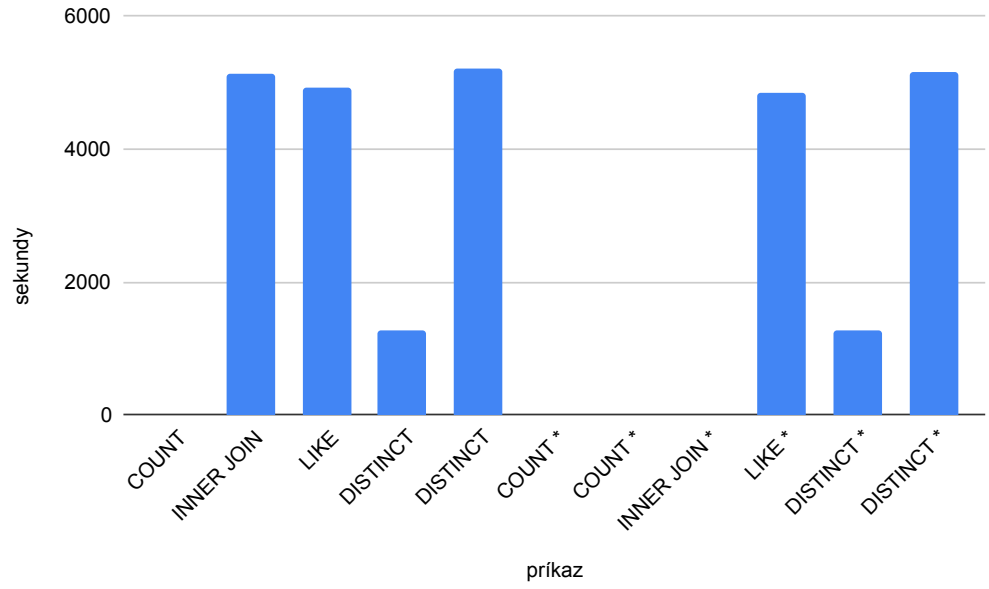
Výsledky záťažových testov



Obr. E.1: Výsledky záťažového testu pre technológiu PostgreSQL



Obr. E.2: Výsledky záťažového testu pre technológiu Greenplum



Obr. E.3: Výsledky záťažového testu pre technológiu MongoDB